SIMULATION-BASED AIRCRAFT ROUTE PLANNING

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

By

ANURAG SINGH

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)

JUNE, 2002

CANDIDATE`S DECLARATION

I hereby certify that the work which is being presented in this project entitled **"Simulation-Based Aircraft Route Planning"** in partial fulfillment of the requirement for the award of the degree of Master of Computer Applications, submitted in the Department of Mathematics of the Indian Institute of Technology, Roorkee, is an authentic record of my work carried out in the period from Jan-2002 to May-2002, under the supervision and guidance of **Dr. R.S. Anand**, Assistant Professor, Department of Electrical Engineering, **Mrs. Aparna Malhotra**, Sc. 'C' , Defense Research & Development Organization.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date : 05.06.2002

Place : IIT, Roorkee.

Anurag Singh

(Anurag Singh)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

[illegible] 5/6/02

Mrs. Aparna Malhotra
Sc. 'C'
Defence R & D Organization
New Delhi

Date:
Place: New Delhi

[illegible]

Dr. R S Anand
Assistant Professor
Department of Electrical Engineering
IIT, Roorkee

Date:
Place: Roorkee

Forwarded
H.G. Sharma

Professor & Head
Department of Mathematics
I.I.T. Roorkee-247 667

11.6.02

DEBASIS DUTTA
Sc 'E'
Head, HRD Cell

No: ISSA/HRD/08
Institute for Systems Studies and Analys
Defence R&D Organisation
Ministry of Defence
Metcalfe House, Delhi - 110 054

Tel: 3818897
Fax: 3819033

Dated: 5 June 2002

CERTIFICATE

This is to certify that **Shri Anurag Singh,** a student of MCA, Indian Institute of Technology Roorkee (6th Semester) has undergone training at ***Institute for Systems Studies & Analyses (ISSA), Defence Research & Development Organisation, Metcalfe House, Delhi –110054*** for a duration of six months commencing from Jan 2002. During the training, he has been involved in the development of the project entitled ***"Simulation Based Aircraft Route Planning".***

He has completed the work assigned to him satisfactorily.

[illegible]

(Debasis Dutta)

प्रणाली अध्ययन एवं विश्लेषण संस्थान
Institute for System Studies & Analys
रक्षा अनुसंधान एवं विकास संगठन
Defence R & D Organisation
रक्षा मन्त्रालय/Ministry of Defen
मेटकॉफ भवन, दिल्ली-110054.
Metcalfe House, Delhi-11005

ACKNOWLEDGEMENT

It is my proud privilege to express my profound gratitude to my guide **Dr. R.S. Anand**, Assistant Professor, Department of Electrical Engineering for his invaluable inspiration, guidance and continuous encouragement throughout this project work.

I am grateful to **Mrs. Aparna Malhotra,** Scientist 'C', Defense Research and Development Organisation and other staff members for providing the necessary facilities, support and inspiration for the successful completion of this work.

I also acknowledge **Mr. Debasis Dutta**, Scientist 'E', **Mr. S.B. Taneja**, Scientist 'D', **Mr. Sanjay Bisht**, Scientist 'B', DRDO for providing the support and inspiration throughout the project work.

I also acknowledge **Dr. H.G. Sharma**, Professor & Head, Department of Mathematics for providing the necessary facilities, cooperation and inspiration.

I would also like to thank **Prof. R.C. Mittal**, Department of Mathematics for his cooperation and support.

In the last but not least I am short of adequate words in expressing thankfulness to my parents and sisters who are the constant source of encouragement to me. It is the only love, care and understanding of my parents that have placed me at the present level of academic career.

[illegible] 11.06.2002.

Anurag Singh

MCA (Final Year)

IIT-Roorkee

ABSTRACT

Real war situation causes much of the destruction of human life and property so there is a need of application that can simulate this type of situation. Simulation methods are used to plan within environments involving large-scale uncertainty, multiple interacting elements and complex dynamics which simulates the actions of agents and intentions of coordinates before committing to a plan.

Simulation Based Aircraft Route Planning application involves multiple interacting intelligent objects, called *agents* and their complex dynamics. The project have been developed in Graphical User Interface (GUI) based scenario editor, with capabilities to add elements, such as Ammunitions Factory of the enemy (Target), Enemy Fighter Aircraft, Ground based Radar units, Surface to Air Missiles (SAM) systems. Also, it provides user to edit attributes of the elements, which are required in simulation process. Further the GUI have capabilities to specify other parameters such as weights attached to cost of route in terms of distance and probability of detection for each route, which plays a role in the simulation activity. Appropriate data structures have been designed to represent the state information as well as the constituent elements of the system. The route planner module of the application plans to create different possible routes. The role of the simulation-based planner in each case is to determine, keeping in view all the available information, the 'best route' or the near optimal path(s) from a starting point to destroy a specified target or objective and return back to base, given a number of certain and uncertain threats to the mission in a defined scenario. The task of finding the near optimal paths can be achieved by several different techniques, one of which is by use of simulation methods. Results of the activities performed by the simulation module and the optimal route have been displayed using the application's GUI.

The work has been developed using Visual C++ with MFC (Microsoft Foundation Classes) and Win32 API on Windows 98 platform.

Table of Contents

CHAPTER - 1

INTRODUCTION

1.1 About the Organisation

DRDO (*Defence Research and Development Organisation*)

Amalgamating Defence Science Organization and some of the technical development establishments established defence Research and Development Organization (DRDO) in 1958. A separate Department of Defence Research and Development was formed in 1980, which now administers DRDO and its 50 laboratories/establishments.

Dr. V.K. Aatre, currently the Scientific Advisor to the Defence Minister and the Secretary to the Indian Government for Defence Research, is the head of DRDO. He renders advice to the Defence Minister and to the organisations in the Ministry of Defence on all scientific and technological aspects of military operations, logistics, weapon system & equipment. In addition, DRDO undertakes research, design & development of weapon systems, equipment, materials and stores.

The Department of Defence Research and Development formulates and executes programs of scientific research, design and development in the fields of relevance to national security leading to the induction of new weapons, platforms and other equipment's required by the Armed Forces. It also functions as the nodal agency for the execution of major development programs of relevance to Defence through integration of research, development, testing and production facilities with the national scientific institutions, public sector undertakings and other agencies.

Research and development activities at DRDO cover important demarcated disciplines like aeronautics, rockets & missiles, electronics & instrumentation, combat vehicles, engineering, naval systems, armament technology including explosives research, terrain research, advanced computing, artificial intelligence, robotics, works study, systems analysis and life sciences including high-altitude agriculture, physiology, food technology and nuclear medicine. In addition to undertaking research and development activities, DRDO also assists the services by rendering technical advice regarding formulation of requirements, evaluation of systems to be acquired, fire and explosive safety and mathematical and statistical analysis of operational problems.

ISSA (*Institute Of Systems Studies and Analyses*)

ISSA is a part of DRDO. It is an inter discplinary Institute in which scientists from various fields like Mathematics, Operations Research, Statistics, Computer Science, Physics,Electronics and Ballistics are employed. Most of these scientists are trained in more than one discipline and have experience of field units in the operational areas and industrial applications. The institute is presently organized in different groups according to work specialization. These team functions as a complete study groups, dedicated to the projects and imparts training to the students of various Universities / Organisations.

It was started as a small group named "Weapon Evaluation Group (WEG)" in the year 1959, primarily to carry out Operational Research (OR) studies and weapon analysis for the three services. In 1963 this group was redesignated as Scientific Evaluation Group and later named as Directorate of Scientific Evaluation (DSE) in 1968. In 1981 its name was changed to Institute of system Studies and Analysis with somewhat broader charter of duties. The Institute is presently organized in different groups as per the expertise.ISSA has to its credit many a succesful projects undertaken by DRDO. To name a few, the wargaming software's being used by Indian Army: Shatranj, Sangram, MIL GIS and Indian Navy : Sagar. It also involved with many projects in Wargaming and Mathematical Modelling and Simulation. It now functions as an establishment similar to other R&D LABS/ESSTS and is located in Metcalfe House Complex.

1.2 About the Project

Simulation based Aircraft Route Planning is an air interdiction application, which builds a Graphical User Interface (GUI) scenario editor which encapsulates different interacting agents actively involved in the interdiction mission. The GUI provides the facility for the deployment of certain and uncertain enemy objects such as

- Radars
- Surface-to-Air missiles
- Enemy Fighters

which pose threat to friendly forces. The GUI provides the facility to depict the area in the form of grid so as to simplify the distance-based computations. Further apart from the default colors used to represent the forces, the GUI provides option to the user to specify the colors they want. The scenario created can be saved to a file, which helps to avoid recreation of the scenario every time simulation is to be done on the same scenario. GUI also facilitates the creation of different candidate routes that are to be simulated.

The main task of the interdiction mission is then to find out the best route from a starting point to a specified target (enemy ammunitions factory) objective, given a number of certain and uncertain threats to the mission. It contributes in the area of planning to develop a method that allows simulation to be used in real time, where the simulation is embedded within the decision-making system.

Application involves the "use of air strike forces to destroy, delay or disrupt existing enemy surface forces while they are far from friendly forces". It involves many interacting elements including concurrently active adversarial tasks and uncertain information regarding ground-based anti-aircraft capability. The application create models for multi-agent systems that are tightly constrained by their environments, i.e.,

- geometric paths that are to be followed
- a task or mission to be accomplished
- a strict set of operating conditions for all agents

The interdiction mission includes attacks against supplies and lines of communication. The objective of the air interdiction mission is to reduce the enemy threat by diminishing enemy combat effectiveness and/or by preventing build up of enemy combat capabilities. To achieve this objective, careful and comprehensive planning is required to isolate an area and to stop all support from reaching the theatre of conflict.

The development of Simulation Based Aircraft Route Planning involves the following phases:

1. Scenario editor / GUI which supports reading, editing and writing scenario definition files
2. Data Structure Design & Text File Format Design module
3. Simulation and Execution Module

CHAPTER - 2

OVERVIEW

2.1 Problem Definition

An Air Interdiction Scenario is basically a combat mission planning where military operations involving two or more opposing forces exercising real life combat rules on data and procedures are involved in the war scenario. The Combat here takes place between the Red force (enemy force) and the Blue force (friendly force).

The mission of the Blue Force Aircraft is to destroy a Red Force Ammunitions Factory to stop all support from reaching the area of conflict. There are a number of certain and uncertain hostile resources such as Radars, Surface-to-air Missiles, each with different detection range and enemy fighters that pose threat to Blue Force Aircraft, whose aim is to destroy or disrupt Red Force Ammunitions Factory and come back to its base safely. At a first glance, the problem of guiding the blue force around the radar, SAM and Aircraft Coverage and toward the factory seems like a simple problem in computational geometry. Infact this is the manner in which most routes planning is done. A typical rule might be found - "to locate a path, avoid radar and SAM fields, and avoid fighting against enemy fighters". However such rule based reasoning becomes more onerous when uncertainty occurs and for the objects in our Air Force Mission Planning domain we have categorize the uncertainty into dynamics are present several types:

- *Uncertainty of existence* : the object may or may not even exist.
- *Uncertainty of location* : An area of uncertainty of the object's location is available but it is not certain of the exact location of the object.
- *Uncertainty of range* : The exact detection range or firing range is not known.
- *Uncertainty of firepower* : The destruction capability of the object is uncertain.

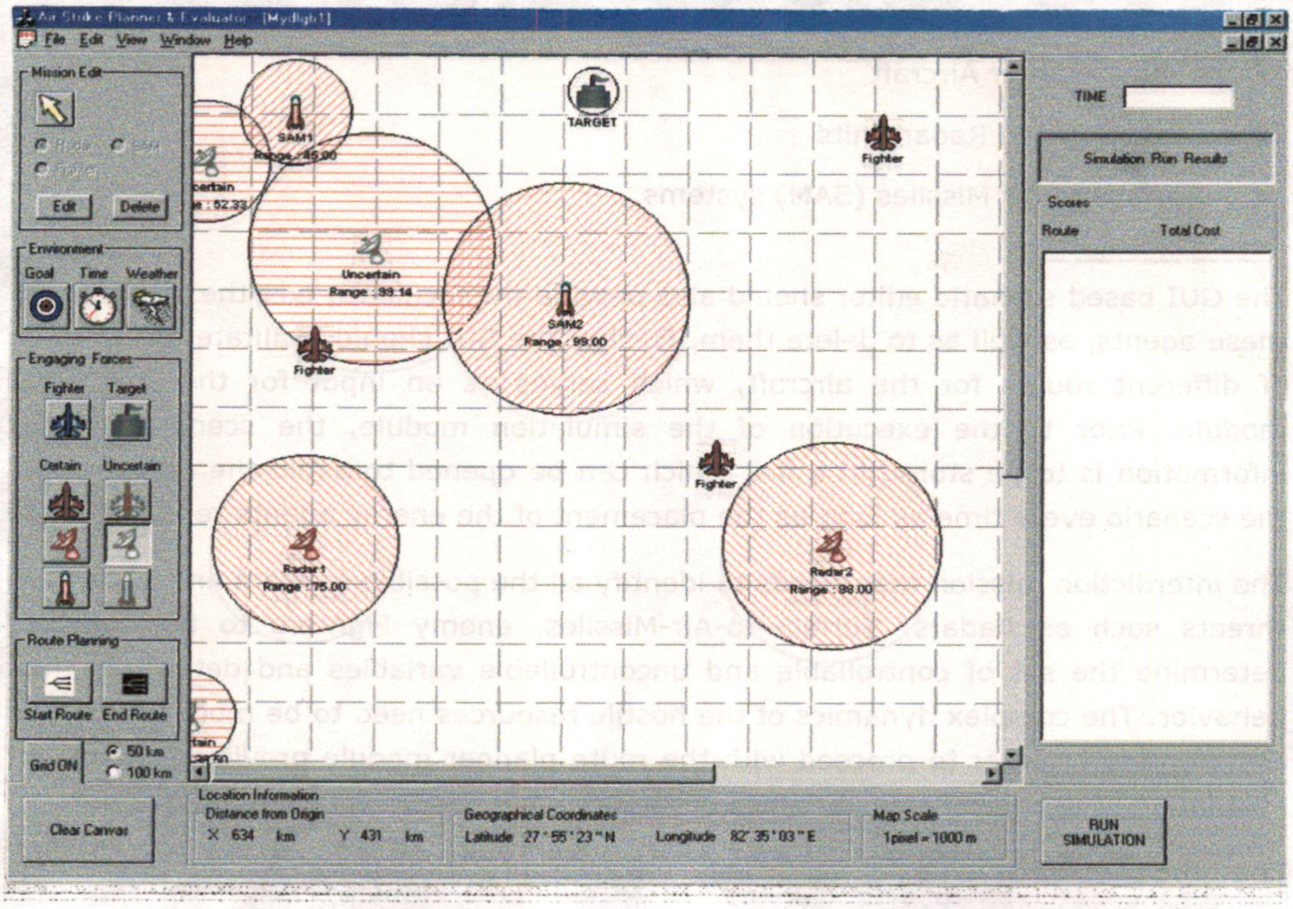


Figure 2.1 A Overview of Scenario (GUI)

Simulation assumes a significant importance as simulation of military operations used for research, analysis, education, or recreation is designed for better understanding of warfare. It involves people in the information processing within an actual or hypothetical situation who use rules, data and procedures to guide various military actions.

2.2 Goals to be achieved

An interdiction mission is a very common problem in defence and its role is to destroy the supply lines of the enemy in order to delay its support. The application first needs to develop a Graphical User Interface (GUI) that creates an interdiction scenario providing facility to place different agents actively participating in the interdiction mission, such as –

- Ammunitions Factory of the enemy (Target)
- Enemy Fighter Aircraft
- Ground based Radar units
- Surface to Air Missiles (SAM) systems,

The GUI based scenario editor should also provide the facility to edit the attributes of these agents, as well as to delete them. Further the GUI should facilitate the creation of different routes for the aircraft, which serves as an input for the simulation module. Prior to the execution of the simulation module, the scenario related information is to be stored in a file, which can be opened to avoid the recreation of the scenario every time as long as the placement of the enemy agents remains fixed.

The interdiction mission then needs to identify all the possible (certain and uncertain) threats such as Radars, Surface-to-Air-Missiles, enemy Fighters to the mission, determine the set of controllable and uncontrollable variables and determine their behavior. The complex dynamics of the hostile resources need to be modeled to build the scenario in order to proceed with the route planner module providing a base for simulation. The controllable variables or factors are sometimes called as parameters.

The primary objective of the Air Interdiction mission entails the execution of carefully conceived, comprehensive plan designed to isolate an area and to stop all support from reaching the area of conflict. Therefore, the task of the Air Strike mission is then to gather the effects of the uncontrollable variables by using random numbers, while varying the parameters to find a near-optimal route which aims at destroying the target (i.e. Red Force ammunitions factory) and a safe return of the Blue Force Fighter to its base. Our aim is not just to find the shortest route from Blue Force base to the target, but the one, which considers various enemy agents to achieve

- Maximum Destruction
- Minimum Threat
- Minimum Cost

CHAPTER - 3

Methodologies Adopted

3.1 Simulation

Computer system users, administrators, and designers usually have a goal of highest performance at lowest cost. Modeling and simulation of system design trade off is good preparation for design and engineering decisions in real world jobs. Simulation is used as a tool to better understand and optimize performance and/or reliability of systems; it is also extensively used to verify the correctness of designs. Most if not all digital integrated circuits manufactured today are first extensively simulated before they are manufactured to identify and correct design errors. Simulation early in the design cycle is important because the cost to repair mistakes increases dramatically the later in the product life cycle that the error is detected [6], [7].

Another important application of simulation is in developing "virtual environments", e.g., for training. Analogous to the holodeck in the popular science-fiction television program Star Trek, simulations generate dynamic environments with which users can interact "as if they were really there." Such simulations are used extensively today to train military personnel for battlefield situations, at a fraction of the cost of running exercises involving real tanks, aircraft, etc.

System Simulation is the mimicking of the operation of a real system, such as the day-to-day operation of a bank, or the value of a stock portfolio over a time period, or the running of an assembly line in a factory, or the staff assignment of a hospital or a security company, in a computer. Instead of building extensive mathematical models by experts, the readily available simulation software has made it possible to model and analyze the operation of a real system by non-experts, who are managers but not programmers. A simulation is the execution of a model, represented by a computer program that gives information about the system being investigated. The simulation approach of analyzing a model is opposed to the analytical approach, where the method of analyzing the system is purely theoretical. As this approach is more reliable, the simulation approach gives more flexibility and convenience. The activities of the model consist of events, which are activated at certain points in time and in this way affect the overall state of the system. The points in time that an event is activated are randomized, so no input from outside the system is required. Events exist autonomously and they are discrete so between the executions of two events nothing happens.

3.1.1 Definition of Simulation

Simulation is the imitation of a real-world process or system overtime. Simulation involves the generation of an artificial history of the system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented. Simulation is an indispensable problem-solving methodology for the solution of many real-world problems. Simulation is used to describe and analyze the behavior of a system, ask what-if questions about the real system, and aid in the design of real systems. Both existing and conceptual systems can be modeled with simulation.

3.1.2 Advantages of simulation

Competition in the computer industry has led to technological breakthroughs that are allowing hardware companies to produce better products continually. It seems that every week another company announces its latest release, each with more options, memory, graphics capabilities, and power what is unique about new developments in the computer industry is that they often act as a springboard for related industries to follow. One industry in particular is the simulation software industry. As computer hardware becomes more powerful, more accurate, faster and easier to use, simulation software does too. Some of the major advantages are:

- *Choose correctly:* Simulation lets us test ever aspects of a proposed change or addition without community resources to their acquisition. This is critical, because once the hard decisions have been made, the bricks have been laid, or the material handling systems have been installed, changes and corrections can be extremely expensive.
- *Compress and expand time:* By compressing and expanding time, simulation allows just speeding up or slowing down phenomenon so that we can investigate them thoroughly.

- *Understand why:* With simulation we can determine the answer to the "why" questions by reconstructing the scene and taking a microscopic examination of the system to determine why the phenomenon occurs.
- *Explore possibilities:* Once we have developed a valid simulation model, we can explore new policies, operating procedures, or methods without the expense and disruption of experimenting with the real system.
- *Diagnose problems:* Simulation allows us to better understand the interacting among the variables that make the complex systems. Diagnosing problems and gaining insight into the importance of these variables increases our understanding of their important effects on the performance of the overall system.
- *Identify Constraints:* By using simulation to perform bottleneck analysis, we can discover the cause of the delays on work in process, information, materials, or other processes.
- *Develop Understanding:* Simulation studies aid in providing understanding about how a system really operates rather than indicating someone's predictions about how a system will operate.
- *Visualize the plan:* Depending on the software used, we might be able to view our operation from various angles and levels of magnification, even in three-dimensional.
- *Build Consensus:* Using simulation to present design changes creates an objective opinion.
- *Prepare for change:* Interacting with all those what-if questions involved in a project during the problem-formulation stage gives us an idea of the scenarios that are of interest.
- *Invest wisely:* The typical cost of a simulation study is substantially less than 10% of the total amount being expended for the implementation of a design or redesign.

- *Train the team:* Simulation models can provide excellent training when designed for that purpose. The team, and individual members of the team, can learn by their mistakes and learn to operate better. This is much less expensive and less disruptive than on-the-job learning.
- *Specify requirements:* The specification for a particular type of machine in a complex system to achieve a desired goal might be unknown. By simulating different capabilities for the machine, the requirements can be established.

3.1.3 Areas of Application

The applications of simulation are very vast. Some of the important applications are:

- *Manufacturing and Material Handling Applications*

  Presentations included the following, among many others:

  -Design and analysis of large-scale material handling systems

  -Analysis of the effects of work-in-process levels on customer satisfaction

  -Assessing the cost of quality

- *Public Systems Applications*

  Presentations included the following, among many others:

  1. *Health Systems*

     -Screening for abdominal arotic aneurysms

     -Lymphocite development in immune-compromised patients

     -Asthma dynamics and medical amelioration

     -Timing of liver transplants

     -Diabetic retinopathy

     -Evaluation of nurse staffing and patient population scenarios

     -Evaluation of automated equipment for a clinical processing laboratory

     -Evaluation of hospital surgical suite and critical care area

  2. *Military Systems*

     -Airforce support equipment use

     -Analysis of material handling equipment for propositioning ships

     -Development and implementation of measures of effectiveness

     -Reengineering traditional stovepipe Army staffs for information operations

-Evaluation of theatre airlift system productivity

-Evaluation of C-1141 depot maintenance

-Evaluation of air mobility command channel cargo system

3. *Natural Resources*

   -Nonpoint-source pollution analysis

   -Weed scouting and weed control decision-making

   -Evaluation of surface water quality data

4. *Public Services*

   -Emergency ambulance system analysis

   -Evaluation of field offices with a government agency

- *Service System Applications*

  Presentations included the following, among many others:

1. *Transportation*

   -Analysis of intelligent vehicle highway systems

   -Evaluation of traffic control procedures at highway work zones

   -Evaluation of taxi management and route control

   -Evaluation of rapid transit modeling with automatic and manual controls

2. *Computer System Performance*

   -User transaction processing behavior analysis

   -Evaluation of database transaction management protocols

   -Evaluation of analytic models of memory queuing

3. *Air transportation*

   -Evaluation of human behavior in aircraft evacuates

   -Analysis of airport/airline operations

4. *Communications Systems*

   -Trunked radio network analysis

   -Evaluation of telephone service provisioning process

   -Picture archiving and communication system analysis

   -Evaluation of modeling of broadband Tele-communication networks

   -Analysis of virtual reality for Tele-communication networks

3.1.4 Military Simulation

Simulation has been applied extensively and successfully to a wide range of military problems, including war gaming, acquisition, logistics, and communications. For e.g., it has been used as a decision support tool to evaluate how a battle force should be constituted, how it might be deployed, and how the weapon systems should be acquired and maintained[2],[5].

Military simulation models are different from others because

- Many of them are highly classified, with details that could not be widely disseminated.
- Weapon capabilities and use aren't typically used in other modeling and simulation (M&S)
- Potential adversaries closely control certain algorithms to avoid reverse engineering.
- The use of certain equations isn't typical of commercial M&S.

Classification of Military Simulation

According to Defense Science Board, military simulations are classified into three categories, i.e., *live, virtual, and constructive*.

Live simulation involves real people and real systems where every move and every shot fired between two groups is monitored by a powerful laser engagement system that records all the signals from the pieces of armor and other equipment that are participating in the exercise. All of this information is fed into the computer simulation, and numerous statistics are tallied so that at the end of the exercise, both teams can be evaluated and areas of improvement can be identified.

Virtual simulation involves real people in a simulated system. This includes aircraft and tank simulators. This type of simulator is helpful in training and in evaluating control, decision, and communication skills. Virtual simulation has become more popular with developments in computer technology, especially computer graphics.

In *constructive* simulation, humans might be (or might not be) interact with model, and everything is simulated. Constructive simulation of combat include war-games for training as well as for analytical tools. Constructive simulation training is usually designed for staff level use and virtual simulation training for operator level use.

3.2 Object-Oriented Methodology

Simulation-based Air force Route Mission Planning is an application, which we have developed in VC++, an object-oriented programming language. An object-oriented programming (OOP) language has many advantages, which can tackle big program very easily.

3.2.1 Benefits of OOP

OOP offers several benefits to both the program designer and the user. Object-Orientation contributes to the solution of many problems associated with the development and quality of software products [3]. The new technology promises

- Greater programmer productivity
- Better quality of software
- Lesser maintenance cost

The principal advantages are:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that can't be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to these objects in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in an implementable form.
- Object- Oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Software that is easy to use is hard to build. It is hoped that the OOP language like VC++ would help manage this problem.

3.2.2 Applications of OOP

The most popular application of OOP, up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using OOP techniques.

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in this type of applications because it can simplify a complex problem. The promising area for application of OOP includes:

- Real-time systems
- Simulation and modeling
- Object-Oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAD/CAD system

It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software systems but also its productivity. Object-Oriented technology is certainly change the way software engineers think, analyze, design and implement systems.

3.2.3 Object-Oriented Programming Approach

One characteristic that is constant in the software industry today is the "change". Change is one of the most critical aspects of software development and management. The impact of these developments is often very extensive and raises a number off issues that must be addressed by the software engineers. Most important among them are:

- Maintainability
- Reusability
- Portability
- Security
- Integrity
- User friendliness of software products

Since the invention of the computer, many programming approaches have been tried. These include techniques such as modular programming, top-down programming, Bottom-top programming and structured programming. The primary motivation in each case has been the concern to handle the increasing complexity of programs that are reliable and maintainable. These techniques became popular among programmers over the last two decades.

With the advent of languages such as C, structured programming became very popular and was the paradigm of the 1980s. Structured programming proved to be a powerful tool that enabled programmer to write moderately complex programs fairly easily. However as the programs grew larger, even the structured approach failed to show the desired results in terms of bug free, easy-to-maintain, and reusable programs.

Object-Oriented Programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts. Also, OOP is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand. This means that an object is considered to be a partitioned area of computer memory that stores data and a set of operations that can access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs without modification. It is a new way of organizing and developing programs and has nothing to do with any particular language. However, not all languages are suitable to implement the OOP concepts easily. Languages that support OOP features include Smalltalk, C++, Ada, VC++, Object Pascal and java.

3.2.4 Object-Oriented Paradigm

The fundamental idea behind Object-Oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an *object*. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from unintentional modification by other functions. OOP allows us to decompose a problem into a number of entities called

Objects and then build data and functions around these entities. The combination of data and functions make up an object.

The data of an object can be accessed only by the functions associated with that object that is member functions. If we want to read a data item in an object, we have to call a member function in the object. It will read the item and return the value. We can't access the data directly. The data is hidden, so it is safe from accidental alteration. Data and its functions are to be encapsulated into a single entity. If we want to modify the data in an object we have to exactly what functions interact with it: the member functions in the object, no other functions can access the data. This simplifies writing, debugging and maintaining the program. However, functions of one object can access the functioned of other object. Some of the features Object-Oriented Paradigm are:

- Emphasis is on data rather than procedure
- Programs are divided into what are known as objects
- Data structures are designed such that they characterize the objects
- Methods that operate on the data of an object are tied together in the data structure
- Data is hidden and can not be accessed by an external functions
- Objects may communicate with each others through functions
- New data and methods can be easily added whenever necessary
- Follows bottom-up approach in program design

3.2.5 Basic Concepts of OOP

It is necessary to understand some of the concepts used extensively in OOP.

Objects and Classes

Objects are the basic runtime entities in an object-oriented system. What kinds of things become objects in OOP? The answer to this is limited only by our imagination, but here are some typical categories to start us thinking:

- **Physical Objects**
  - Automobiles in a traffic-flow simulation
  - Electrical components in a circuit design program
  - Aircraft in an air-traffic-control system

- **Elements of the computer-user environment**
    - Windows
    - Menus
    - Graphics objects (lines, rectangles, circles)
    - The mouse and the keyboard
- **Programming constructs**
    - Customize arrays
    - Stacks
    - Linked-Lists
    - Binary trees
- **Connections of data**
    - An inventory
    - A personnel file
    - A dictionary
    - A table of latitudes and longitudes of world cities
- **User-defined data types**
    - Time
    - Angles
    - Complex numbers
    - Points on the plane
- **Components in computer games**
    - Ghosts in a maze game
    - Positions in a board games (chess, checkers)
    - Animals in an ecological simulation
    - Opponents and friends in adventure games

Program objects should be chosen such that they match closely with the real –world objects. When a program is executed the objects interact by sending messages to one another. For example 'customer' and 'account' are two objects in a banking program, then the customer object may send a message to the account object requesting for the balance. Each object contains data and code to manipulate the data. Objects can interact without having to know the details of each other data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects.

Data Abstraction and Encapsulation

The wrapping up of data and functions into a single unit (called class) is known as *encapsulation*. Data Encapsulation is the most striking features of a class. The data isn't accessible to the outside world and only those methods, which are wrapped in the class, can access it. These methods provide the interface between the objects data and the program. This insulation of the data from direct access by the program is called data hiding. Encapsulation makes it possible for objects to be treated like 'black boxes', each performing a specific task without any concern for internal implementation.

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, and cost and functions that operate on these attributes. They encapsulate all the essential properties of the objects that are to be created.

Inheritance

Inheritance is the process by which of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. For e.g., the class of animals is divided into mammals, amphibians, insects, birds, and so on. The class of vehicles is divided into cars, trucks, buses, and motorcycles.

The principle in this sort of division is that each subclass shares common characteristics with the class from which it's derived. Cars, trucks, buses and motorcycles all have wheels and a motor; these are the defining characteristics of vehicles. In addition to the characteristics shared with other members of the class, each subclass also has its own particular characteristics: buses, for instance, have seats for many people, while trucks have space for hauling heavy loads.

In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. Thus the real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what one wants, and to tailor the class in such a way that it does not introduce any undesirable side effects into the rest of the classes.

Polymorphism

Polymorphism is another OOP concept. Polymorphism (one thing with several distinct forms) means the ability for a new object to implement the base functionality of a parent object in a new way. An operation may exhibit different behavior in different instances. The behavior depends upon the types of data used in the operation.

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accesses in the dame manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing Inheritance.

Dynamic Binding

Binding refers to the linking of a procedural call to the code to be executed in response to the call. Dynamic Binding means that the code associated with a given procedure call isn't known until the time of the call at runtime. It is associated with polymorphism and inheritance. A procedure call associated with a polymorphic reference depends on the dynamic type of that reference.

Message Communication

An Object-Oriented program consists of a set of objects that communicate with each other. The process of programming on an Object-Oriented language, therefore, involves the following basic steps:

1. Creating classes that define objects and their behavior
2. Creating objects from class definitions
3. Establishing communication among objects

Objects communicate with one another by sending and receiving information much the same way as people pass message to one another. A message for an object is a request for execution of a procedure, and therefore, will invoke a procedure in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the method (message) and the information to be sent. For e.g., consider the statement

```
radars. display (position);
```

Here, radars is the object, display is the message and position is the parameter that contains information.

3.3 Software Engineering Methodology

The various phases of software life cycle are:

1. Requirements Analysis
2. Software design
3. Coding
4. Testing
5. Maintenance

Requirement Analysis

Requirement Analysis is done in order to understand the problem, which the software system is to solve. The emphasis is on identifying what is needed from the system and not how the system will achieve its goals. There are two parties involved in software development a client and a developer. The developer has to develop the system to satisfy the client's needs. The developer usually does not understand the client's problem domain, and the client often does not understand the issues involved in software systems. This causes a communication gap, which has to be adequately bridged during requirements analysis.

In most software projects, the requirements phase ends with a document describing all the requirements. Hence the goal of requirements phase is to produce a software requirements specification document. The person responsible for the requirements analysis is often called the analyst. There are two activities in this phase:

1. Analysis or problem understanding
2. Requirements specification

In problem analysis, the analyst has to understand the problem and its context.
Once the problem is analyzed and the essential understood, the requirements must be specified in the requirement specification document.

Software Design

The purpose of the design phase is to plan a solution of the problem specified by the requirements document. This phase is the first step in moving from the problem domain to the solution domain. The output of this stage is the design document. This document is a plan for the solution and is used later during implementation, testing and maintenance.

The design phase has two separate activities:

1. System design-top level design
2. Detailed design

During design phase, two separate documents are produced: one for system design the other for detailed design. Together these specify the design. That is, they specify the models and the internal logic of each of the modules.

Coding

The goal of the coding phase is to translate the design of the system in a given program language. For a given design, the aim in this phase is to implement the design in the best possible manner. The coding phase comes before testing and maintenance. Since the testing and maintenance costs of say much higher than the *coding cost, the goal of coding should be to reduce testing and maintenance effort.*

Testing

Testing is the major quality control measure employed during development. Its basic function is to detect errors in the software. After coding phase, computer programs are available that are executed for testing. Testing uncovers errors introduced in requirement, design, coding programs. Consequently different levels of testing are employed.

The first level of testing is unit testing. In this a module is tested and is often performed by the coder himself simultaneously with the coding module. The aim is to execute the different parts of the module coding and coding errors. After this the modules are integrated into subprograms then integrated themselves to eventually form the actual system programs.

During integration of modules, integration testing is performed. This testing is to detect design errors, while focusing on testing the interdependency between the modules. After the system is put together, system testing is that, the system is tested against the system requirement, to see if all the requirements are met and the system performs as specified by the system requirements. Finally, acceptance testing is performed to demonstrate to the client, on the real life data of the client, the operation of the system.

Maintenance

Maintenance is not a part of software development, but is an extremely important activity in the life of a software product. Maintenance includes all the activities after the installation of software that is performed to keep the system operational. The four major forms of maintenance activities are adaptive maintenance, corrective maintenance, perfective maintenance and preventive maintenance.

CHAPTER - 4

Problem-Solving

4.1 Agent as a Basic Unit

The Air Force is expanding its modeling and simulation activities as a practical solution to improve readiness and lower costs. Modeling and simulation save millions of dollars by cutting the need to deploy actual forces and equipment, as in the case of command and control exercises. Not only American forces, but also those of other countries, can be included in simulated conflicts. The Air Force has identified several desired capabilities that encompass both improvements in existing models and the development of new types of modeling. To satisfy these needs, scientists at the Human Effectiveness Directorate are conducting research to discover efficient ways to simulate intelligent behavior in existing and new models.

Furthermore, Air Force requirements call for the development of completely new capabilities, such as models capable of accurately simulating the conduct of information operations. Information operations mean the execution of information warfare (i.e., an activity designed to manipulate, degrade, deny, or destroy information) with the goal of influencing an opponent battle staff or civilian authority's ability to make decisions.

The basic unit of Simulation is an agent. An agent is *"any actor in a system, any entity that can generate events that affect it and other agents"*. Agents define the basic objects in the system, the simulated components. Simulations consist of groups of many interacting agents. For example, an ecosystem simulation could consist of agents representing coyotes, rabbits, and carrots. In an economic simulation, agents could be companies, stockbrokers, shareholders, and a central bank.

4.2 Decision Making And Planning

Decision making and planning are critical operations for all military missions. Moreover, planning occurs over several different time scales depending on the amount of time one has to plan prior to committing to a particular plan. Planning is a hierarchical enterprise. Our long term goal is to explore this hierarchy of planning approaches, and our first step towards this goal is to provide high-level planner with a technique we call *Simulation Based Planning (SBP)*. Military mission involve many interacting elements including concurrently active adversarial tasks and uncertain information regarding ground-based anti-aircraft actability. As the complexity of the mission increases, it is valuable to use computer simulation. Hence we introduce simulation-based planning as a methodology for addressing the complexity involved in Air Force missions.

4.3 Simulation Based Planning

Simulation Based Planning extends and improves the planning horizon in three aspects. First, it handles probalistic uncertainty through detailed and replicated simulation of models instead of solving them analytically using probability theory. Second, simulation can naturally extend the level of execution and thereby often discovering subtleties (which would have been missed by a higher lever planner) that may lead to failure of a plan. Finally, a multi-agent adversarial planning is easily achieved through object-oriented multimode simulation where each agent or adversary is simulated [1].

Simulation is defined as *"the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer and analyzing the execution output"*.

Once the plan is chosen for execution, the simulation data that was generated during the planning process can be used to match with the current real world state.

Our focus is near-optimal route planning. Route planning is in-between the higher level of symbolic AI planning and the lower level of intelligent control. There are several application areas that are related to route planning. Mission planning within the military domain almost always involves route planning. Routes can greatly affect the success of the whole mission, whether the mission takes place on ground or in the air. If the goal is to select a route that is shortest in distance, we can use any of the standard algorithms that exist for finding the shortest paths in the graph. But, if the problem is in an environment that is unknown or uncertain, we must use different ways to evaluate each path such as simulation.

4.4 Basic Components Of Simulation-Based Route Planning System

The simulation based route planner has three main components: -

1. The Route generator
2. The Simulator
3. The Plan Evaluator/Selector

The Route Generator : The route generating component gathers information about the source and the destination in order to generate candidate routes between them linking source to the target.

The Simulator : The main task of the simulator module is to simulate each candidate route taking into account all the possible threats . The simulation monitors various controllable and uncontrollable factors to analyze each route and generates a score for each route.

The Evaluator : The evaluator mainly evaluates the results of the simulation and then selects the best route amongst the given candidate routes for execution.

4.5 SBP Framework

We present SBP framework in terms of three components :

1. The Simulation component
2. The Experimental Design Component
3. The output Analysis Component

1. *Simulation Block*

To use simulation in planning we, first need to identify the set of controllable and uncontrollable variables. Speeds, routes, actions of objects are controllable, whereas any kind of uncertainty such as uncertainty of positions conditions, range of radars and outcome of combat are uncontrollable. The controllable variables or factors are sometimes called parameters in simulation process. The main objective of the plan simulation is to gather the effects of the uncontrollable through randomness while varying the parameters to find a near-optimal combination of controllable values in spite of the uncertainty. We say near optimal because we can never guarantee the optimality of a plan given the uncertainties of actual plan execution.

In addition to parameters there are artificial factors such as simulation specific variables like the initial state of the system, termination conditions and random number streams. Due to nature of our problem, the user gives artificial factors such as initial state of the environment and termination conditions of plans.

We now define the following:

- Let W = W_1, W_2,, ... Wk be the set of all objects in the environment. And let Q (t) = q_1 (t) * q_2 (t) * ...* q_k(t) be the world state at time t where qi(t) is the state of object Wi at time t , a finite set of world states. Also we define Ai(t) as the set of actions Wi can take at time t and zero or more actions may be chosen from this set to be simulated at time t.
- Let R be set of routes that need to be simulated and chosen from. The total number of routes is N and Rj denotes the jth route where 1 < j< N.
- **Stationary Object** refers to an object that remains physically in the same location throughout the simulation and objects that do not have the ability to physically change location. Ground radars, missile sites are some examples.
- **Moving Objects** refers to the object that has the ability to physically move and change its location during simulation such as fighter.
- **Planner Object** refers to the object, which is planning entity itself.
- Let O be the set of uncertain stationary objects. Then, these objects can have one or more of the following uncertainties:
  - -Initial location
  - -Type (e.g. type of plane, type of missile)
  - -Configuration (e.g. speed, range)

Then the simulation algorithm follows:

- Determine the score for each route R_I, comprising of –
  a. Cost of each route in terms of distance
  b. Detection probability
- For each route Rj,

```
While (there are more routes)
    Get the cost and detection probability of the route
    Calculate the score for the route
```

- Find the route with the smallest cost.

2. *Experimental Design Block*

In simulation, experimental design is a method of choosing which configurations (i.e. parameter values) to simulate so that the desired information can be acquired with the least amount of simulating. In experimental design terminology, the input parameters and structural assumptions composing a model are called *factors* and the output performance measures are called *responses*. The route simulation algorithm models two factors - cost of the route in terms of distance that directly affects the fuel consumption and the probability of friendly aircraft being detected by the enemy units. Thus the objective function is given as: -

Route [i] = w1 * route cost [i] + w2 * route prob [i]

Where,

i = Route No.

Route [i] = Total cost of the route i

route cost [i] = Total cost of route i in terms of distance

route prob [i] = Total probability of being detected on route i

w1 = Weight attached to cost of route i

w2 = Weight attached to probability of detection for route i

3. *Analyzer*

Based upon the selection criteria (i.e. simulation factors) we consider several different plans/routes and choose the best plan for execution. The tasks of this component are to analyze the results obtained from alternate routes and select the one with the minimum score. The selection criteria largely depends upon the weights attached to the factors associated with the route. Assignment of different weights to factors could lead to selection of a different route each time, despite the fact that their scores remain the same.

CHAPTER - 5

Logical Design

5.1 Data Flow Diagram (DFDs)

5.1.1 Context Diagram

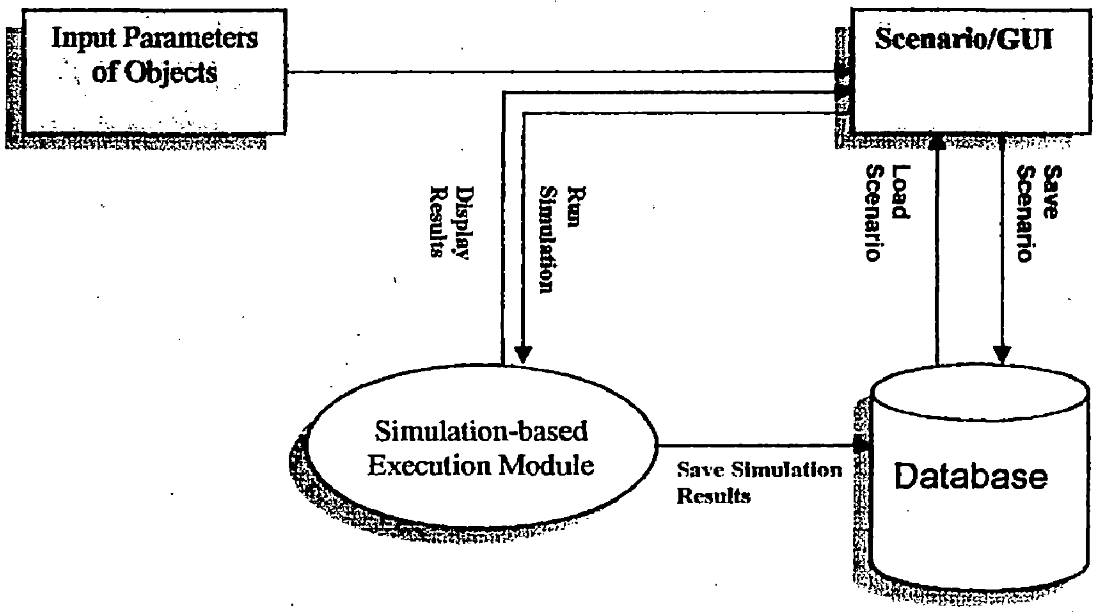


Figure 5.1 Context Diagram

5.1.2 Top Level DFD

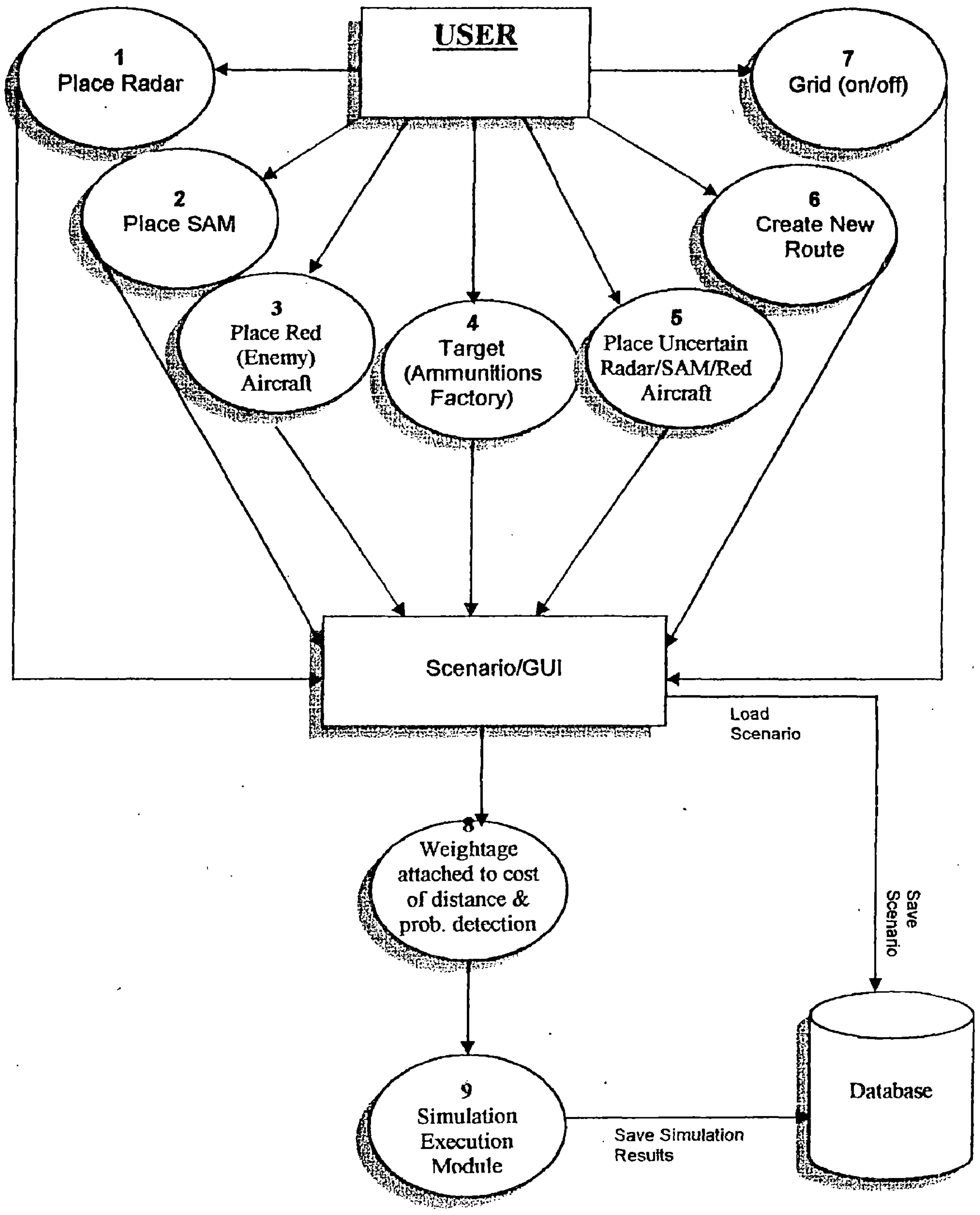

USER
1
Place Radar
7
Grid (on/off)
2
Place SAM
6
Create New Route
3
Place Red (Enemy) Aircraft
4
Target (Ammunitions Factory)
5
Place Uncertain Radar/SAM/Red Aircraft
Scenario/GUI
Load Scenario
Save Scenario
8
Weightage attached to cost of distance & prob. detection
9
Simulation Execution Module
Save Simulation Results
Database

Figure 5.2 Top Level Data Flow Diagram

5.1.3 Data Flow Diagram Level-2

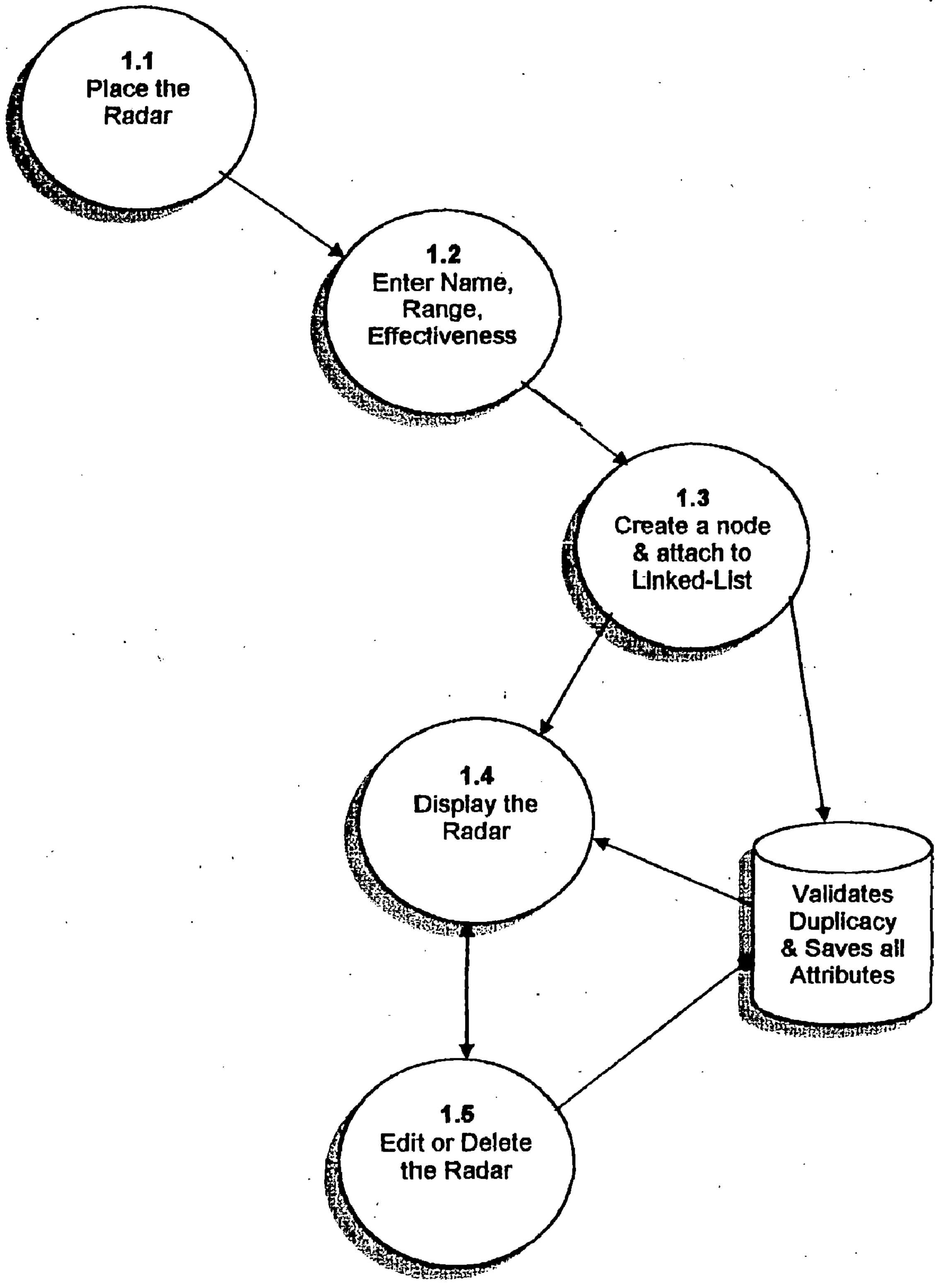

1.1
Place the Radar
1.2
Enter Name, Range, Effectiveness
1.3
Create a node & attach to Linked-List
1.4
Display the Radar
Validates Duplicacy & Saves all Attributes
1.5
Edit or Delete the Radar

Figure 5.3 Detailed DFD for radar operation

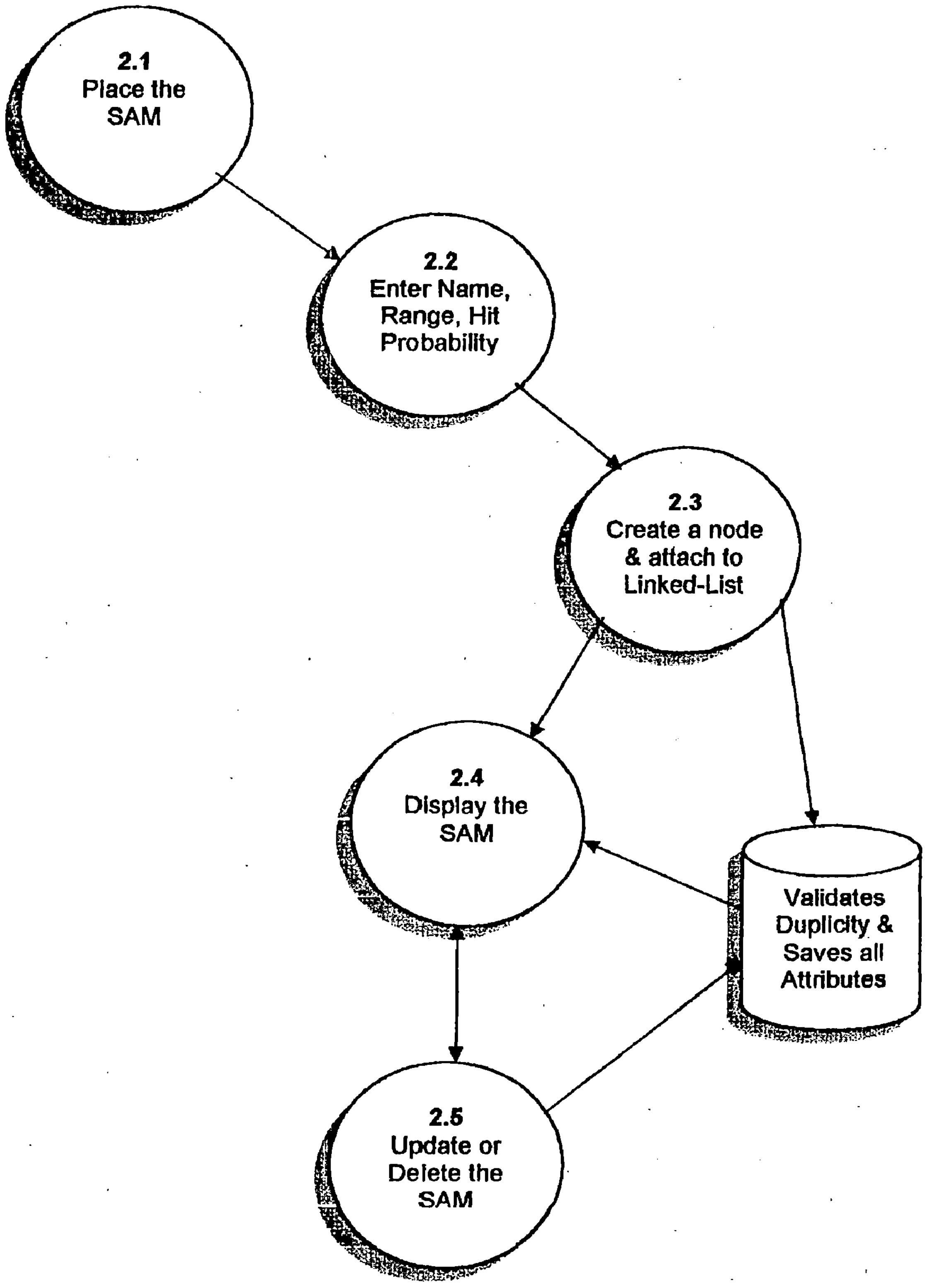

2.1
Place the SAM
2.2
Enter Name, Range, Hit Probability
2.3
Create a node & attach to Linked-List
2.4
Display the SAM
Validates Duplicity & Saves all Attributes
2.5
Update or Delete the SAM

Figure 5.4 Detailed DFD for SAM operation

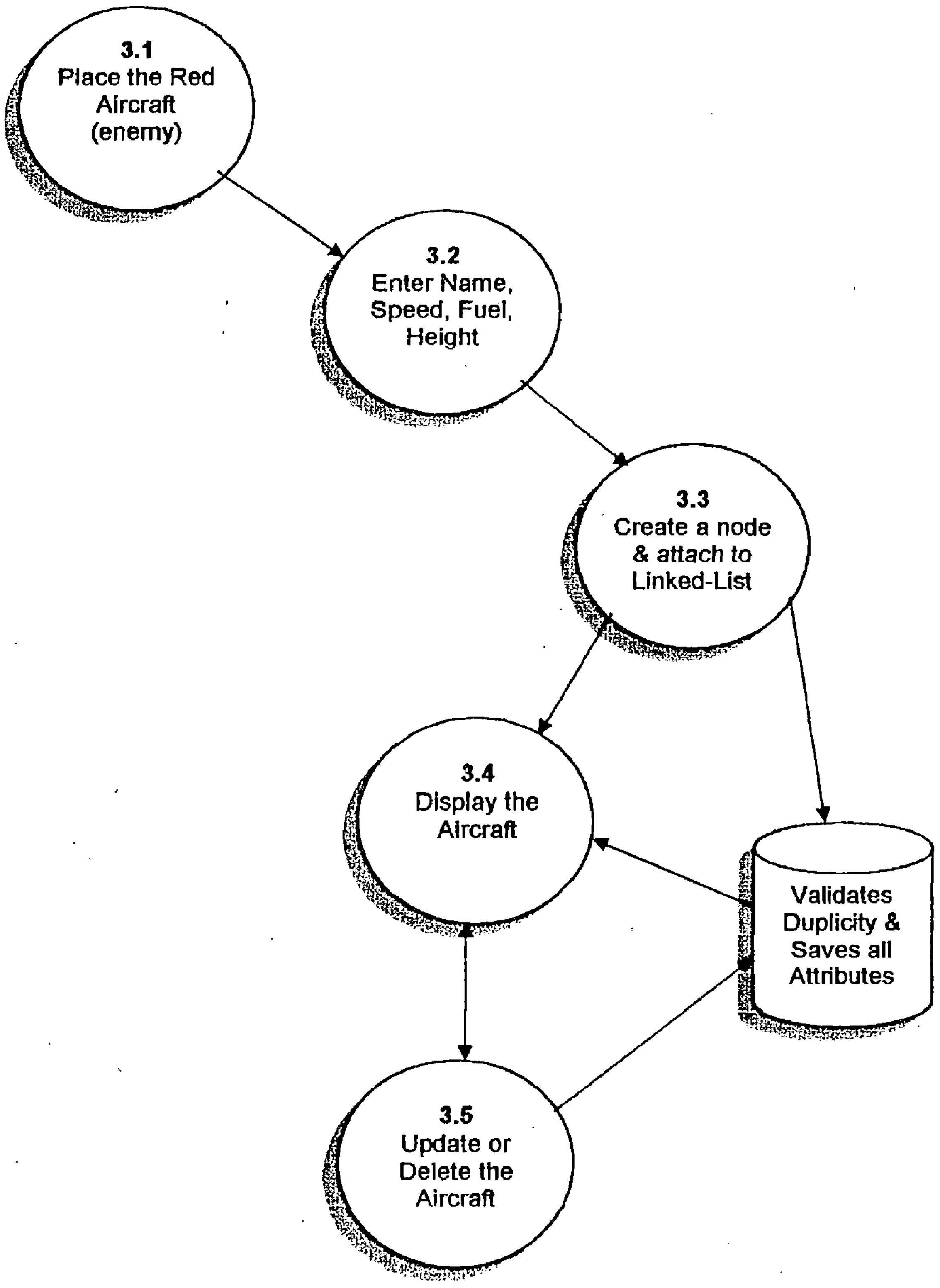

3.1
Place the Red Aircraft (enemy)
3.2
Enter Name, Speed, Fuel, Height
3.3
Create a node & attach to Linked-List
3.4
Display the Aircraft
Validates Duplicity & Saves all Attributes
3.5
Update or Delete the Aircraft

Figure 5.5 Detailed DFD for enemy aircraft operation

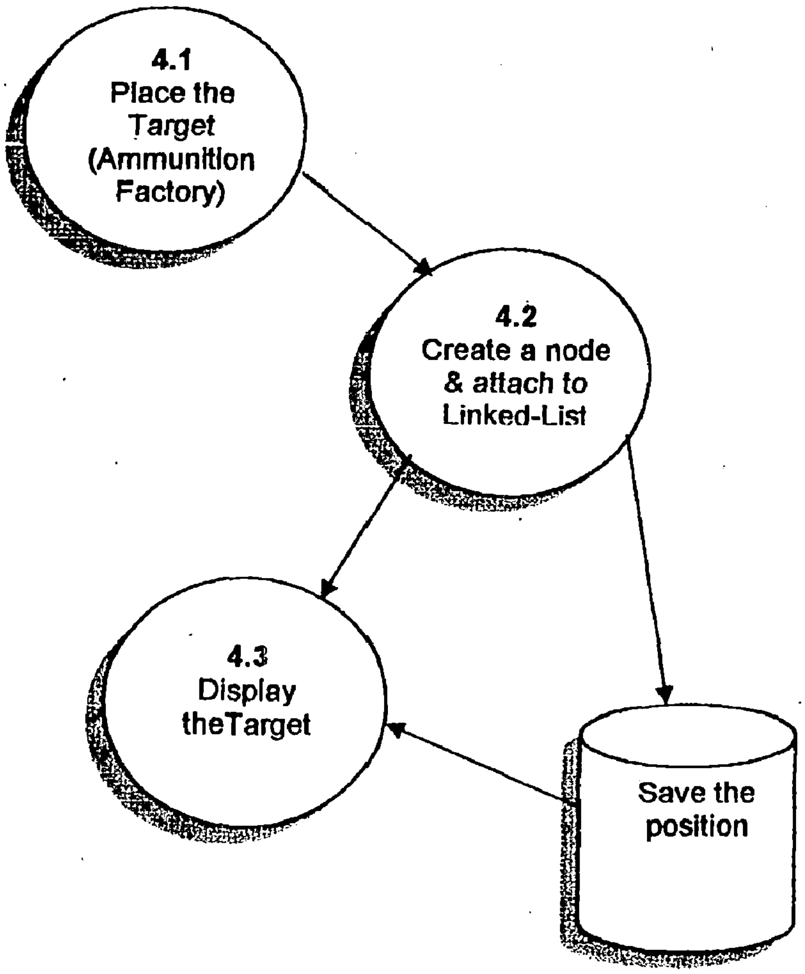

4.1
Place the Target (Ammunition Factory)
4.2
Create a node & attach to Linked-List
4.3
Display theTarget
Save the position

Figure 5.6 Detailed DFD for target operation

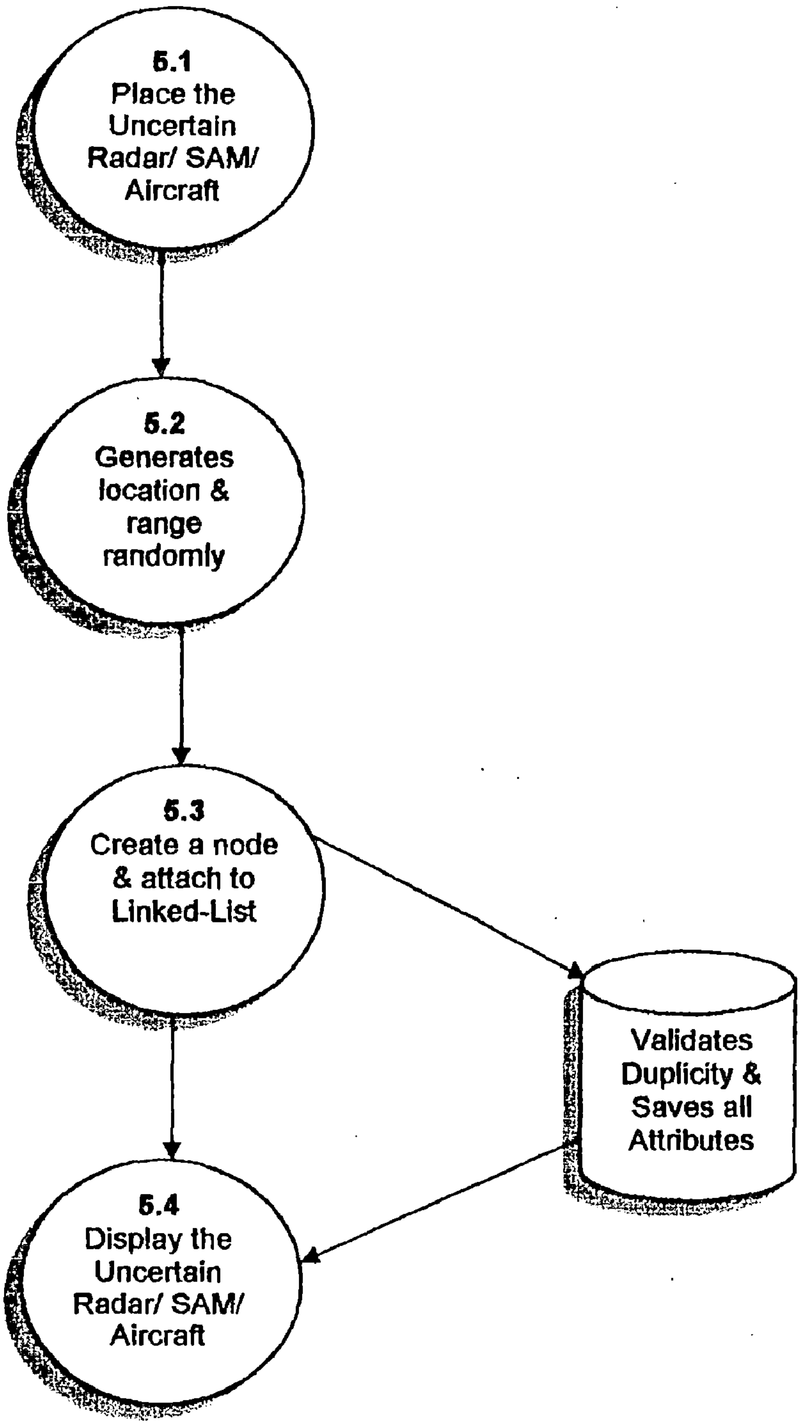

5.1
Place the Uncertain Radar/ SAM/ Aircraft
5.2
Generates location & range randomly
5.3
Create a node & attach to Linked-List
Validates Duplicity & Saves all Attributes
5.4
Display the Uncertain Radar/ SAM/ Aircraft

Figure 5.7 Detailed DFD for uncertain radar/Sam/Aircraft operation

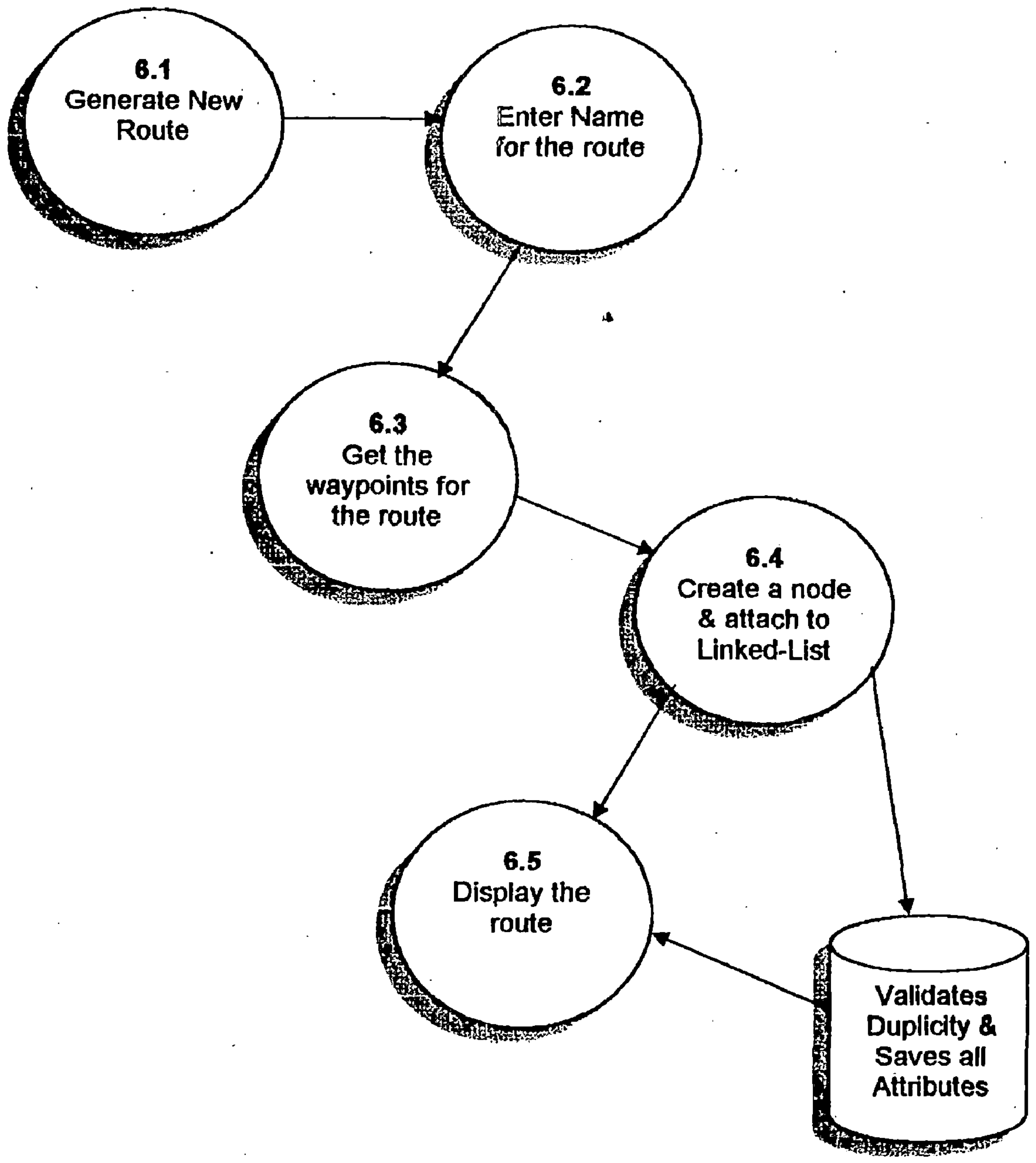

6.1
Generate New Route
6.2
Enter Name for the route
6.3
Get the waypoints for the route
6.4
Create a node & attach to Linked-List
6.5
Display the route
Validates Duplicity & Saves all Attributes

Figure 5.8 Detailed DFD for route operation

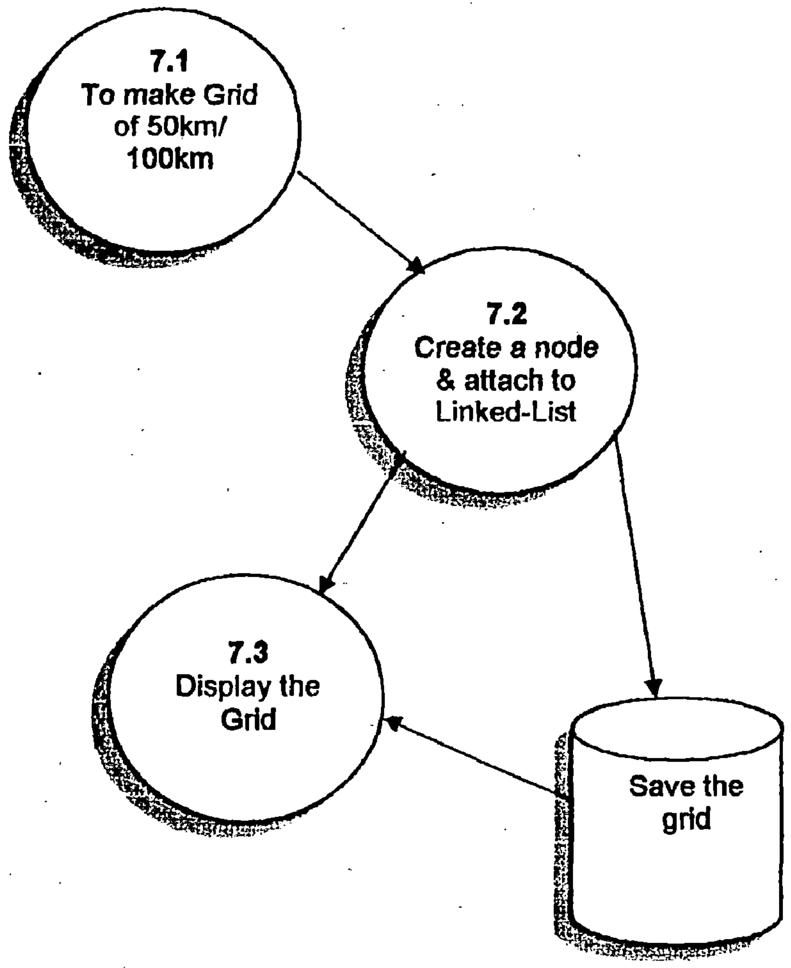

7.1
To make Grid of 50km/ 100km
7.2
Create a node & attach to Linked-List
7.3
Display the Grid
Save the grid

Figure 5.9 Detailed DFD for Grid Formation

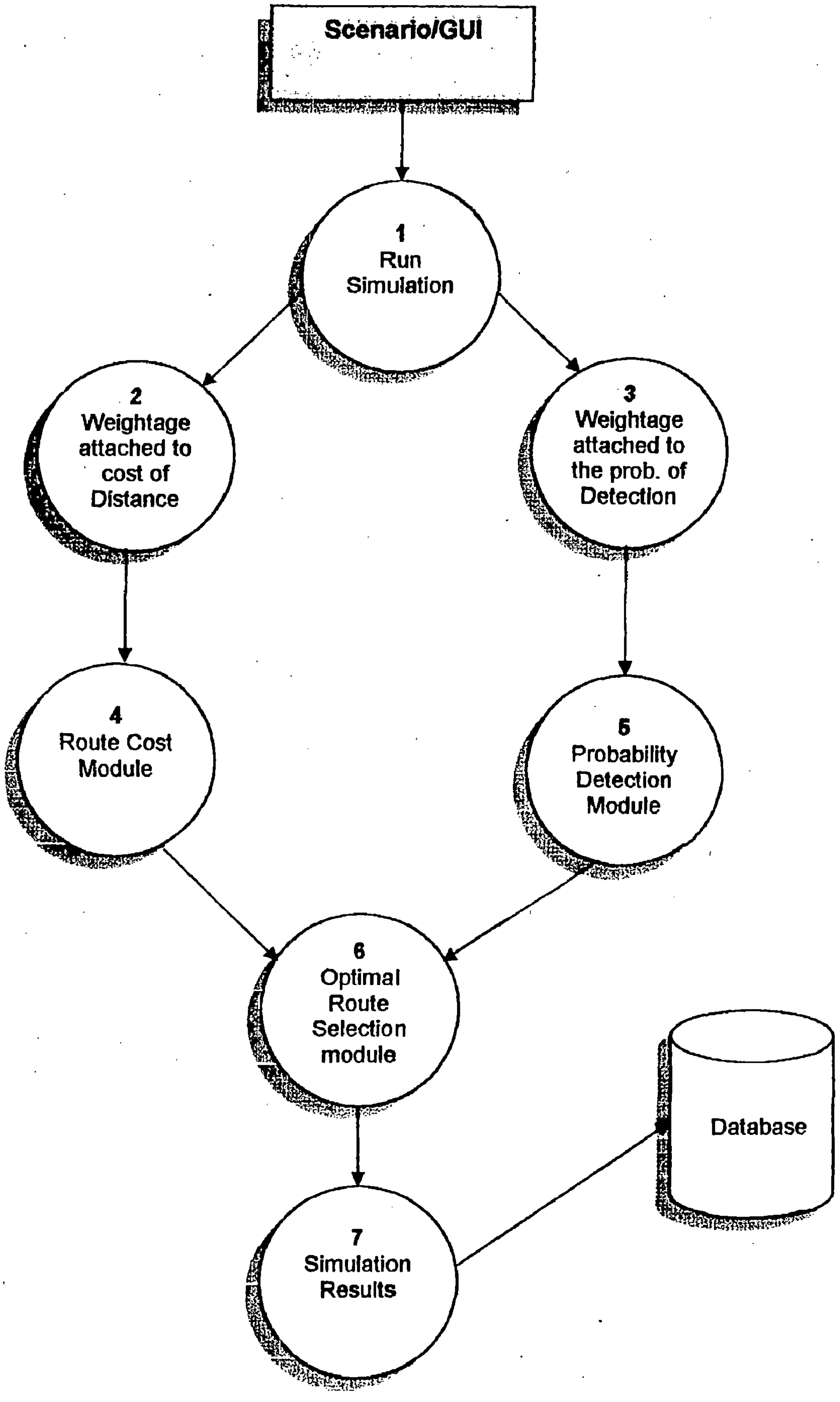
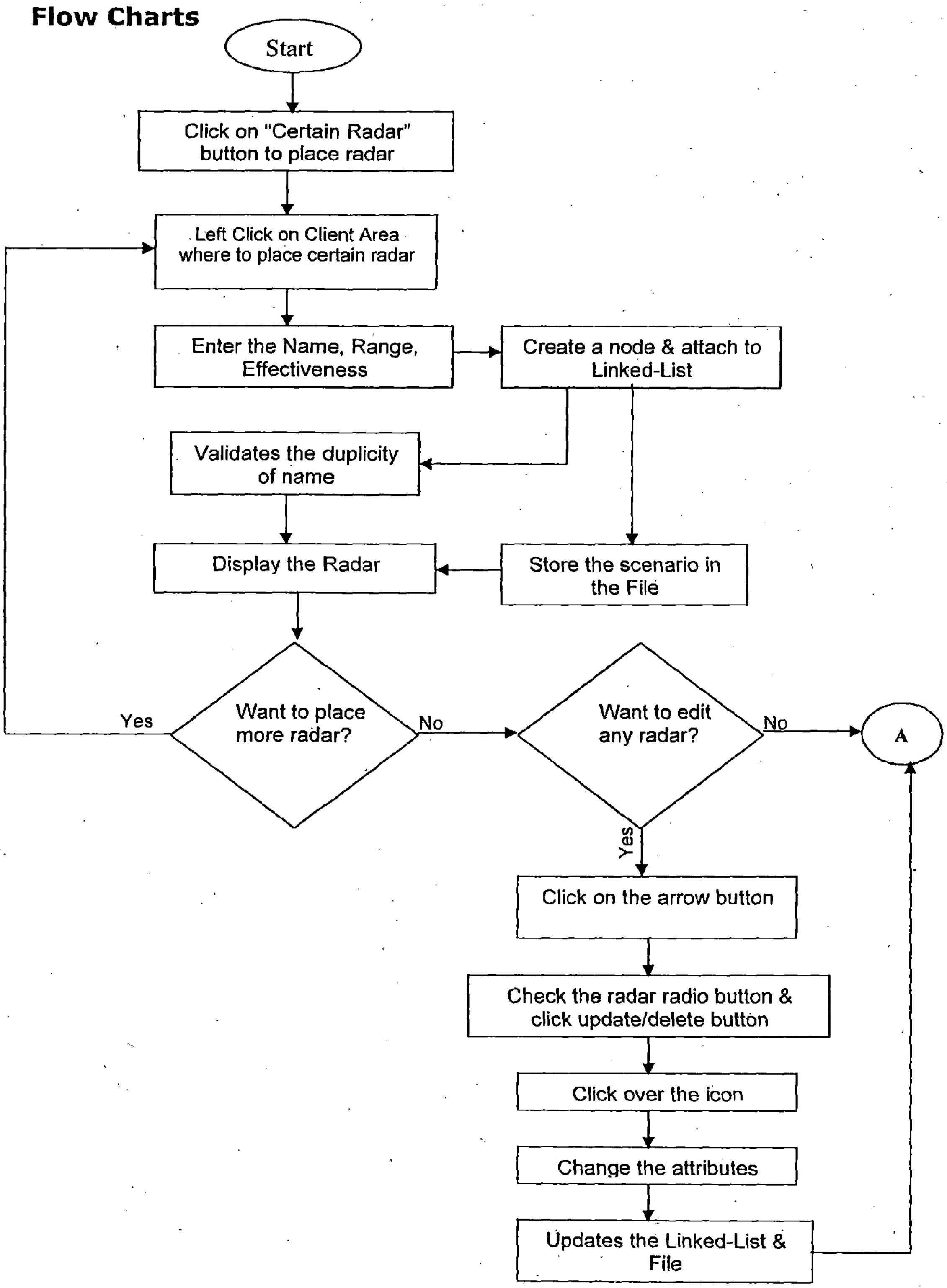

Scenario/GUI
1
Run Simulation
2
Weightage attached to cost of Distance
3
Weightage attached to the prob. of Detection
4
Route Cost Module
5
Probability Detection Module
6
Optimal Route Selection module
7
Simulation Results
Database

Figure 5.10 Level-2 DFD for Simulation Module

5.2 Flow Charts



Start
Click on "Certain Radar" button to place radar
Left Click on Client Area where to place certain radar
Enter the Name, Range, Effectiveness
Create a node & attach to Linked-List
Validates the duplicity of name
Display the Radar
Store the scenario in the File
Want to place more radar?
Yes
No
Want to edit any radar?
No
A
Yes
Click on the arrow button
Check the radar radio button & click update/delete button
Click over the icon
Change the attributes
Updates the Linked-List & File

Figure 5.11 Flow Chart for radar operation

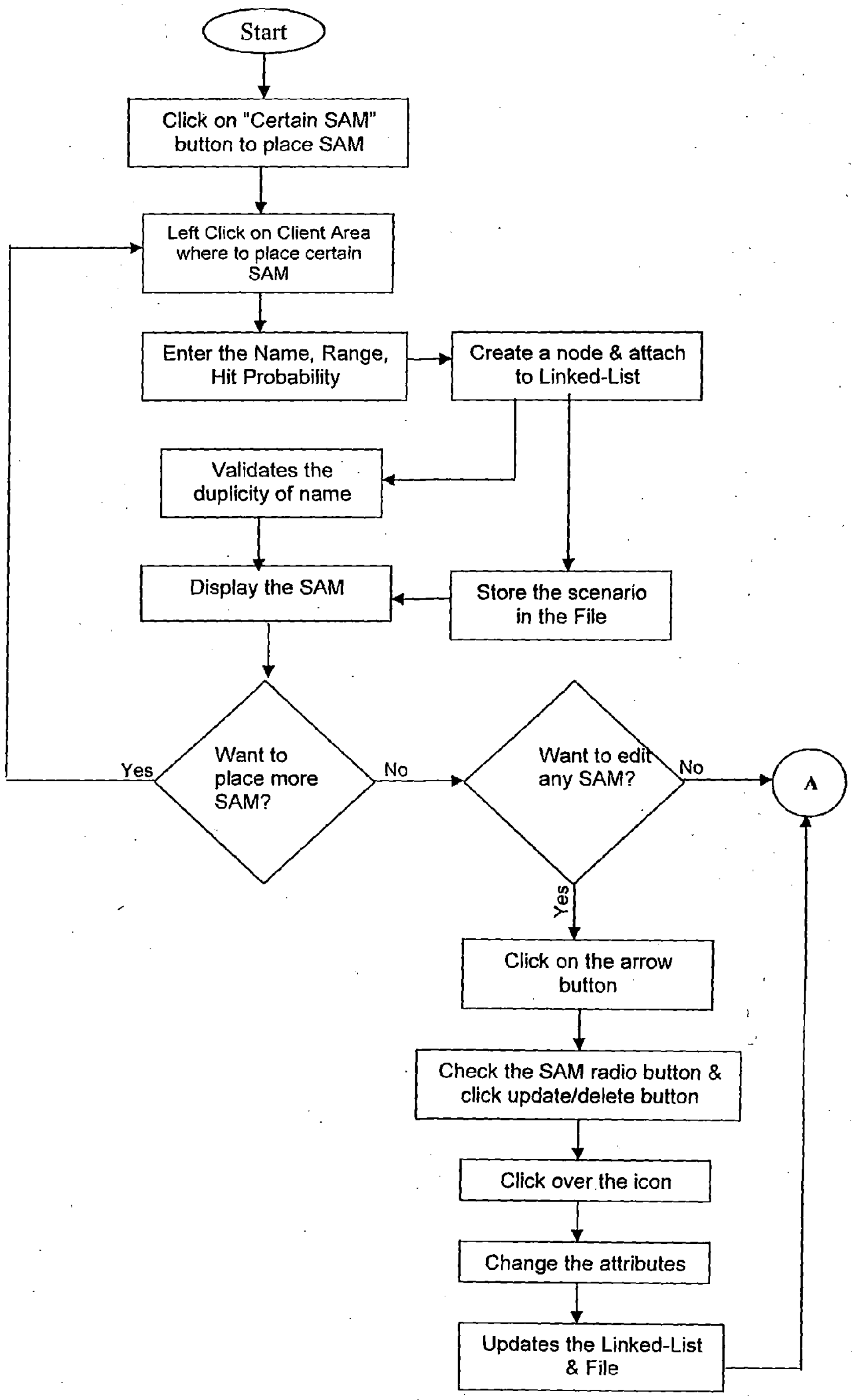
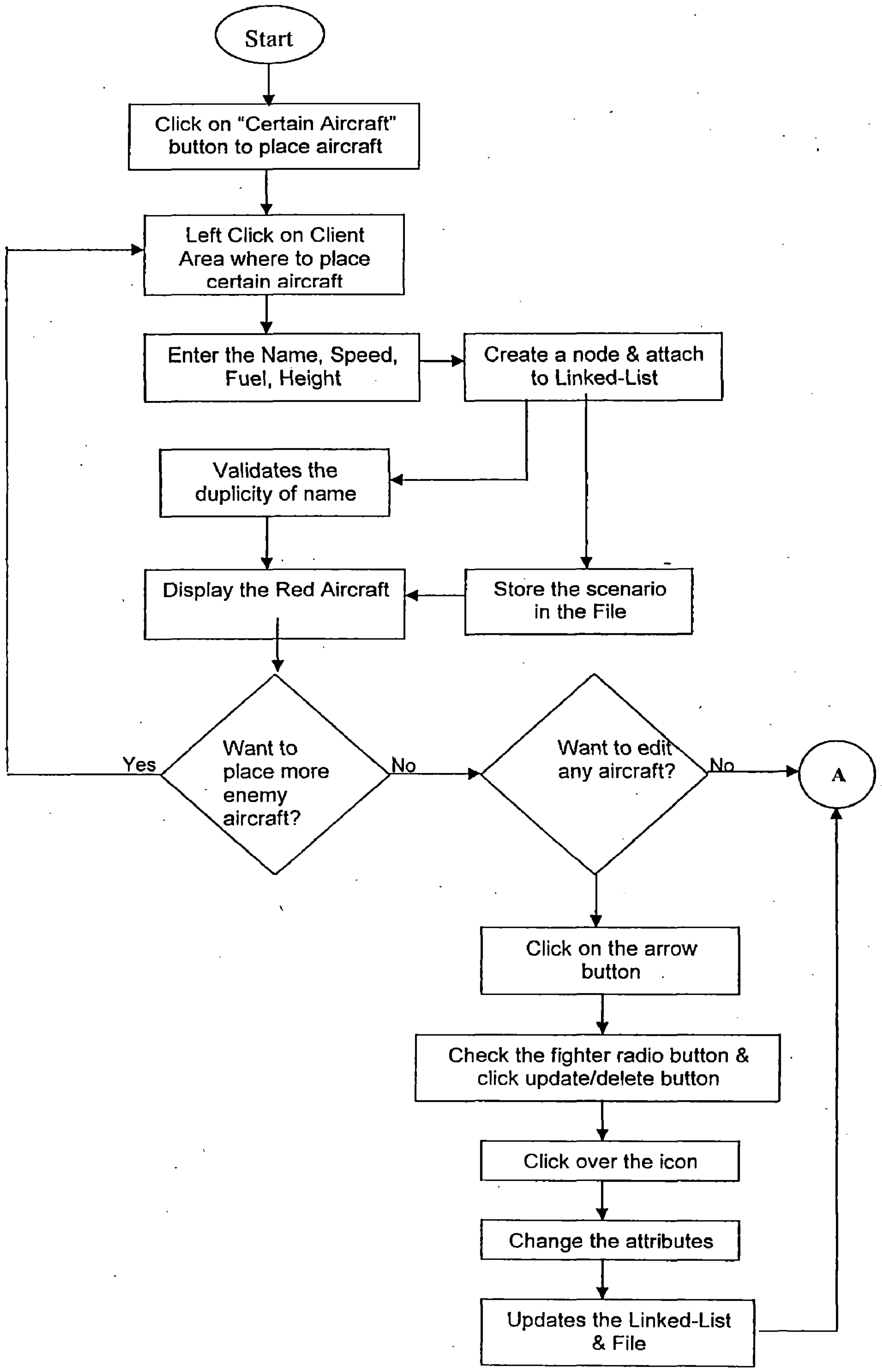

Start
Click on "Certain SAM" button to place SAM
Left Click on Client Area where to place certain SAM
Enter the Name, Range, Hit Probability
Create a node & attach to Linked-List
Validates the duplicity of name
Display the SAM
Store the scenario in the File
Want to place more SAM?
Yes
No
Want to edit any SAM?
No
A
Yes
Click on the arrow button
Check the SAM radio button & click update/delete button
Click over the icon
Change the attributes
Updates the Linked-List & File

Figure 5.12 Flow Chart for SAM operation



Start
Click on "Certain Aircraft" button to place aircraft
Left Click on Client Area where to place certain aircraft
Enter the Name, Speed, Fuel, Height
Create a node & attach to Linked-List
Validates the duplicity of name
Display the Red Aircraft
Store the scenario in the File
Want to place more enemy aircraft?
Yes
No
Want to edit any aircraft?
No
A
Click on the arrow button
Check the fighter radio button & click update/delete button
Click over the icon
Change the attributes
Updates the Linked-List & File

Figure 5.13 Flow Chart for enemy aircraft operation

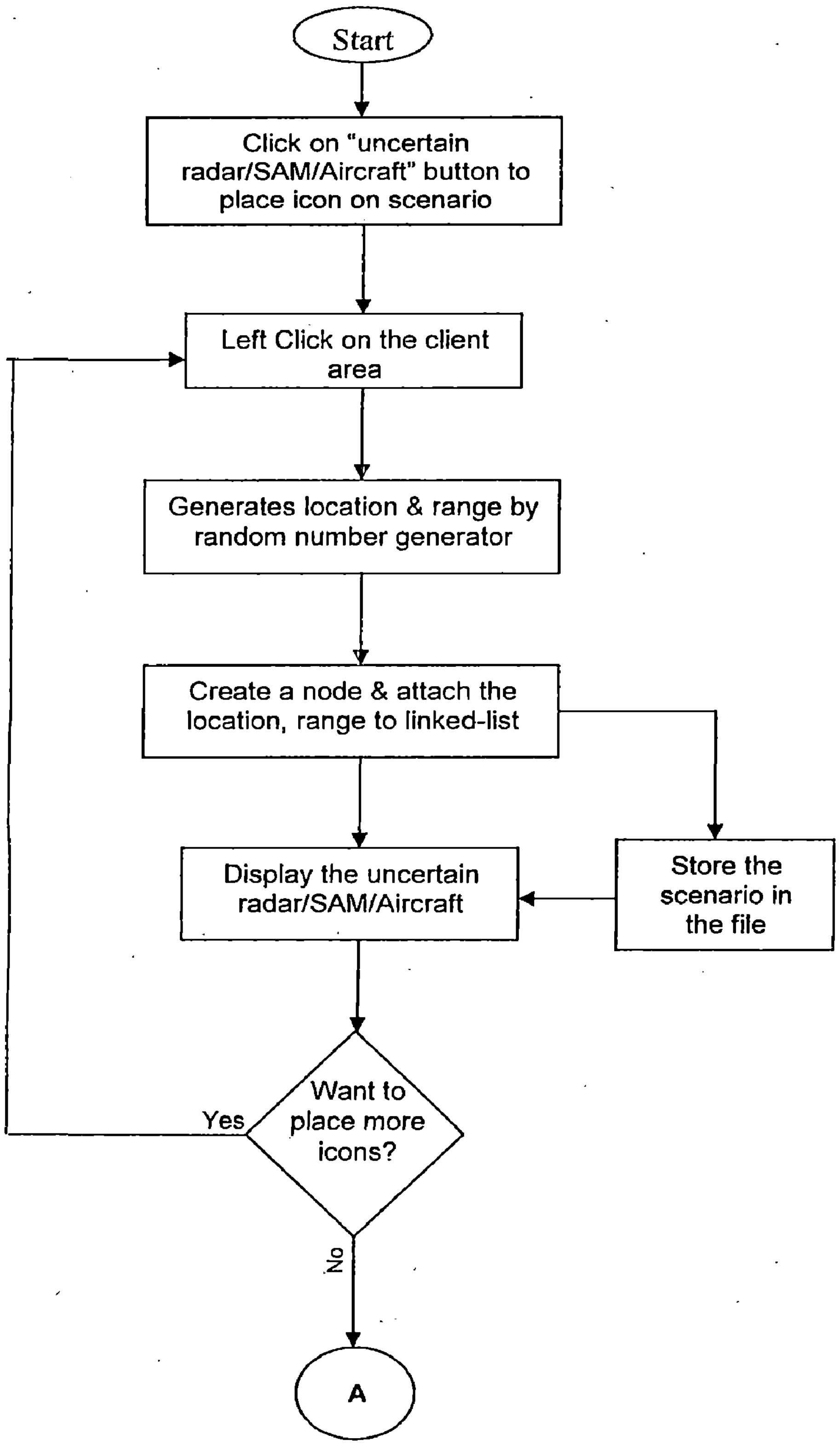

Start
Click on "uncertain radar/SAM/Aircraft" button to place icon on scenario
Left Click on the client area
Generates location & range by random number generator
Create a node & attach the location, range to linked-list
Store the scenario in the file
Display the uncertain radar/SAM/Aircraft
Want to place more icons?
Yes
No
A

Figure 5.14 Flow Chart for Uncertain radar/SAM/Aircraft operation

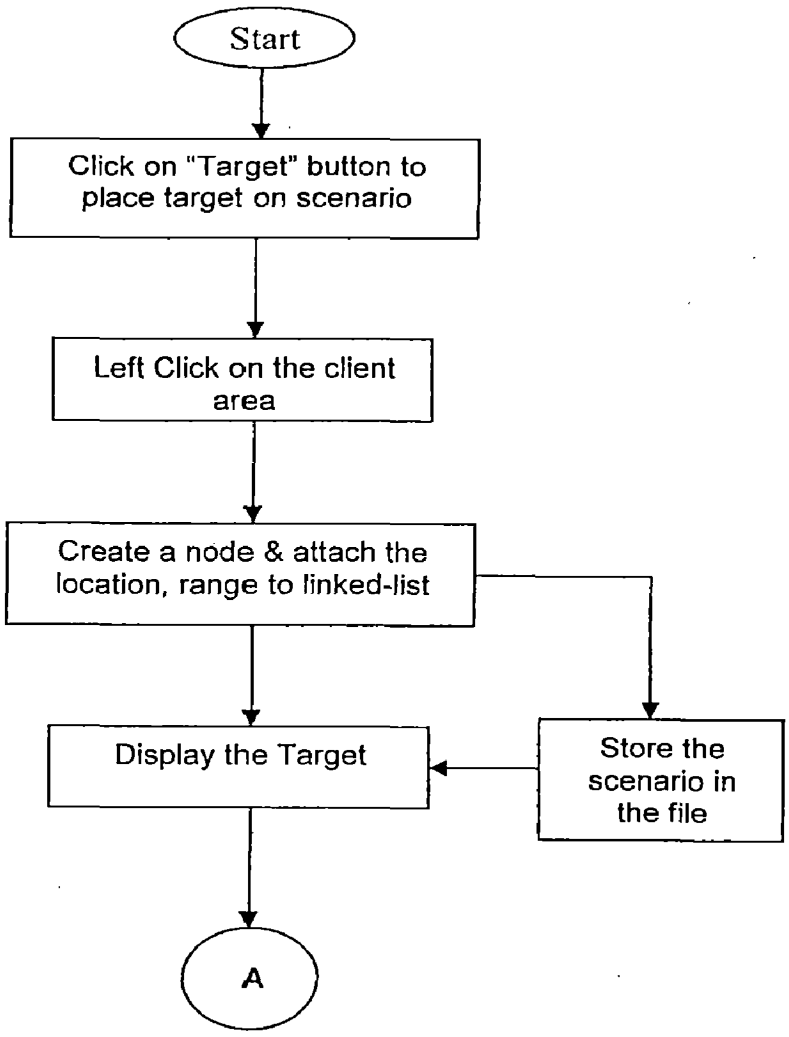
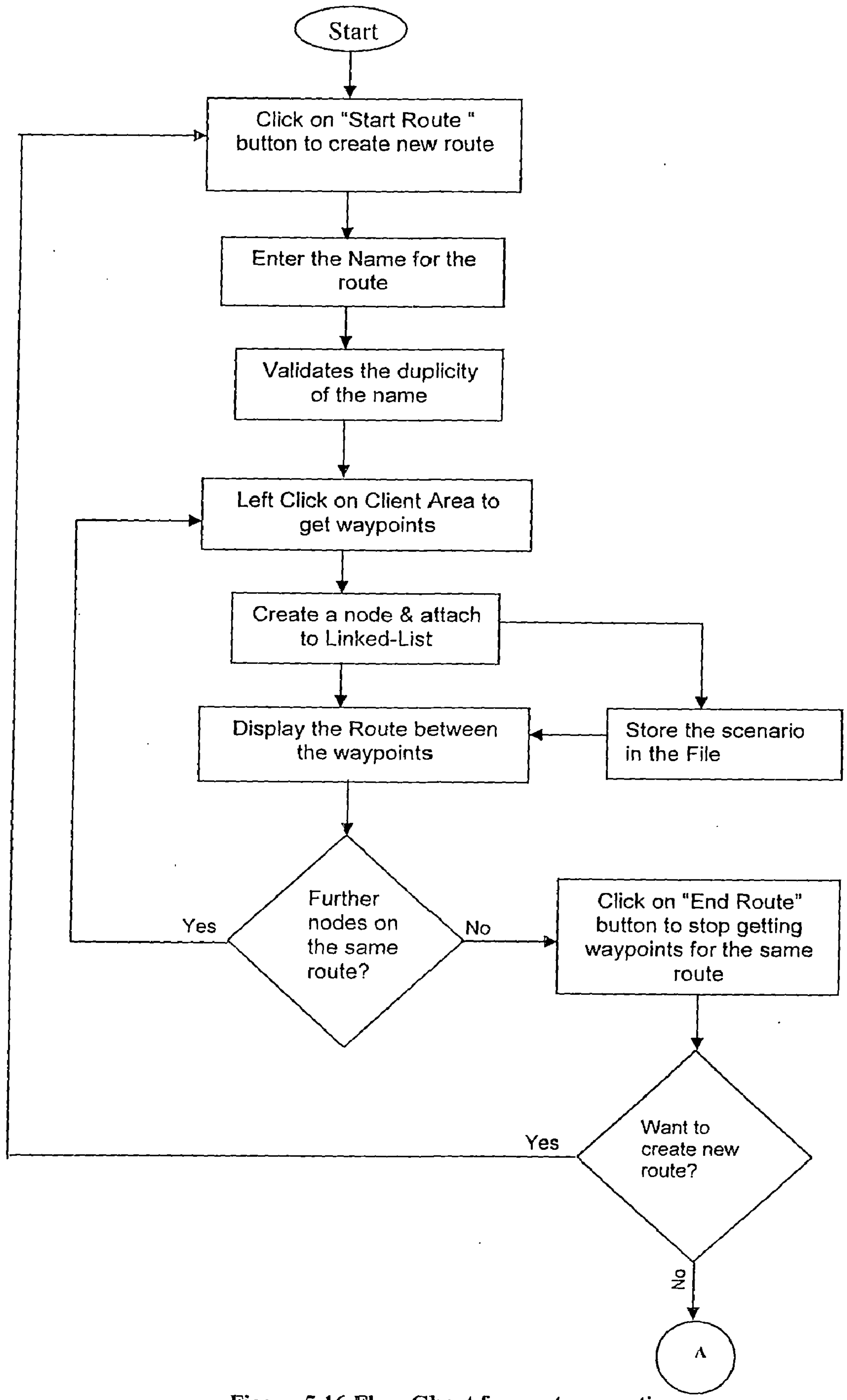

Start
Click on "Target" button to place target on scenario
Left Click on the client area
Create a node & attach the location, range to linked-list
Display the Target
Store the scenario in the file
A

Figure 5.15 Flow Chart for Target Placing



Start
Click on "Start Route " button to create new route
Enter the Name for the route
Validates the duplicity of the name
Left Click on Client Area to get waypoints
Create a node & attach to Linked-List
Display the Route between the waypoints
Store the scenario in the File
Further nodes on the same route?
Yes
No
Click on "End Route" button to stop getting waypoints for the same route
Want to create new route?
Yes
No
A

Figure 5.16 Flow Chart for route operation

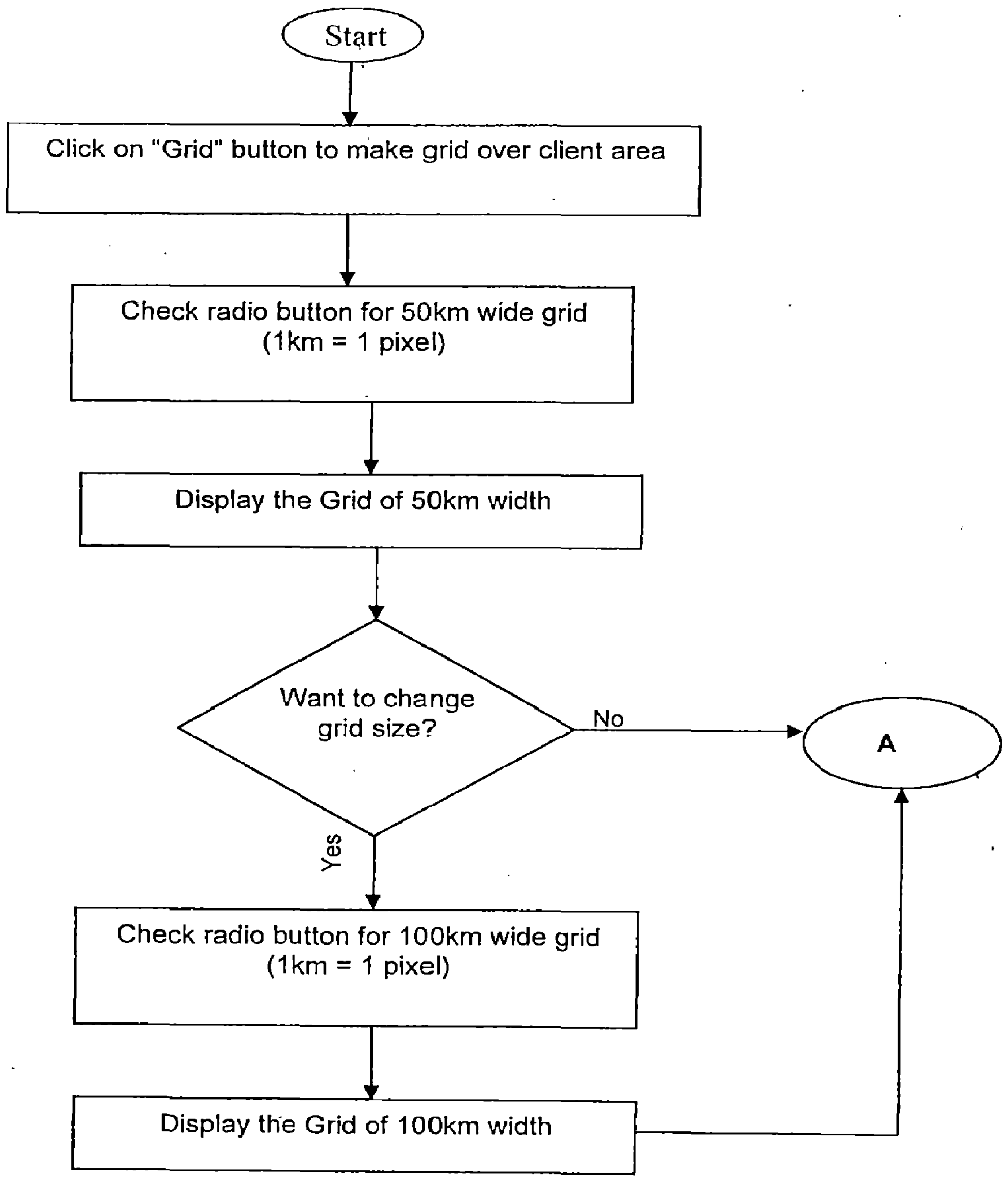

Start
Click on "Grid" button to make grid over client area
Check radio button for 50km wide grid (1km = 1 pixel)
Display the Grid of 50km width
Want to change grid size?
No
A
Yes
Check radio button for 100km wide grid (1km = 1 pixel)
Display the Grid of 100km width

Figure 5.17 Flow Chart for Grid Formation

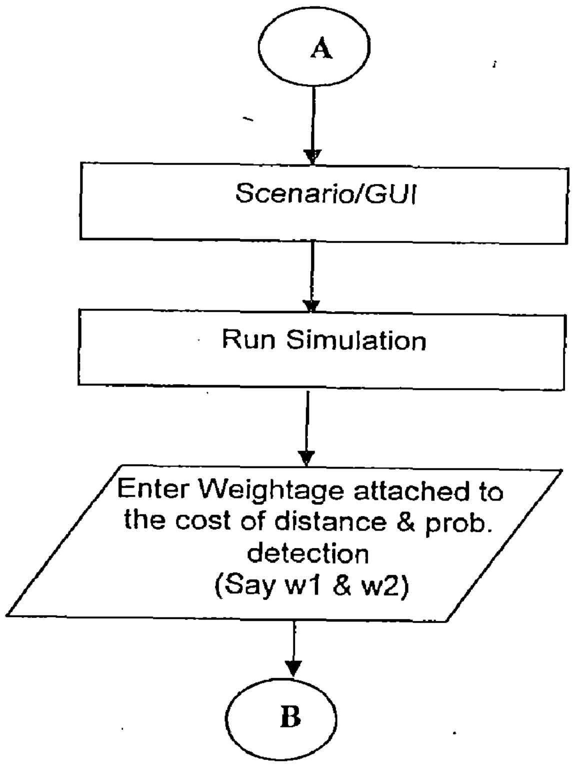

A
Scenario/GUI
Run Simulation
Enter Weightage attached to the cost of distance & prob. detection (Say w1 & w2)
B

Figure 5.18(a) Run Simulation & Assign Weightage

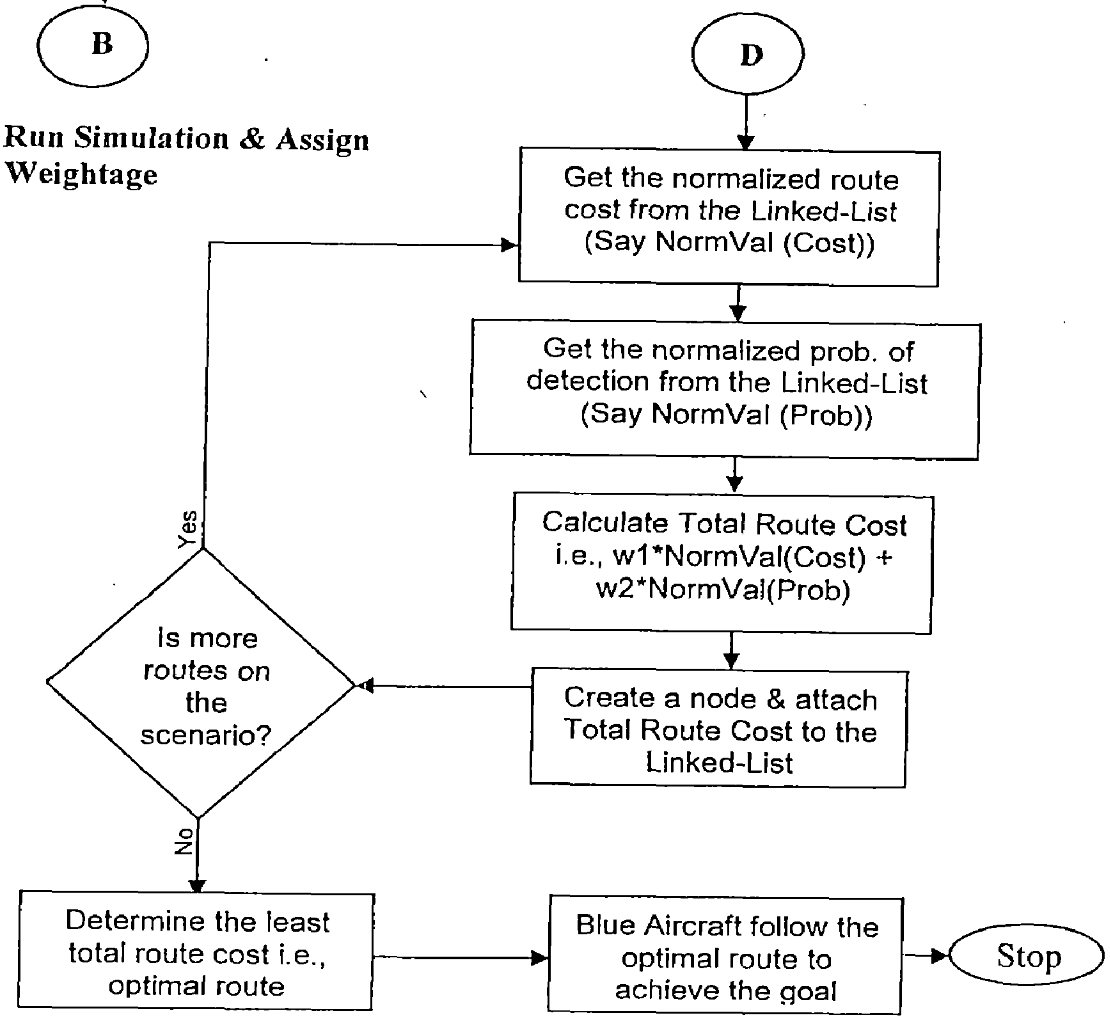

D
Get the normalized route cost from the Linked-List (Say NormVal (Cost))
Get the normalized prob. of detection from the Linked-List (Say NormVal (Prob))
Calculate Total Route Cost i.e., w1*NormVal(Cost) + w2*NormVal(Prob)
Create a node & attach Total Route Cost to the Linked-List
Is more routes on the scenario?
Yes
No
Determine the least total route cost i.e., optimal route
Blue Aircraft follow the optimal route to achieve the goal
Stop

Figure 5.18(b) Flow Chart for Simulation Module

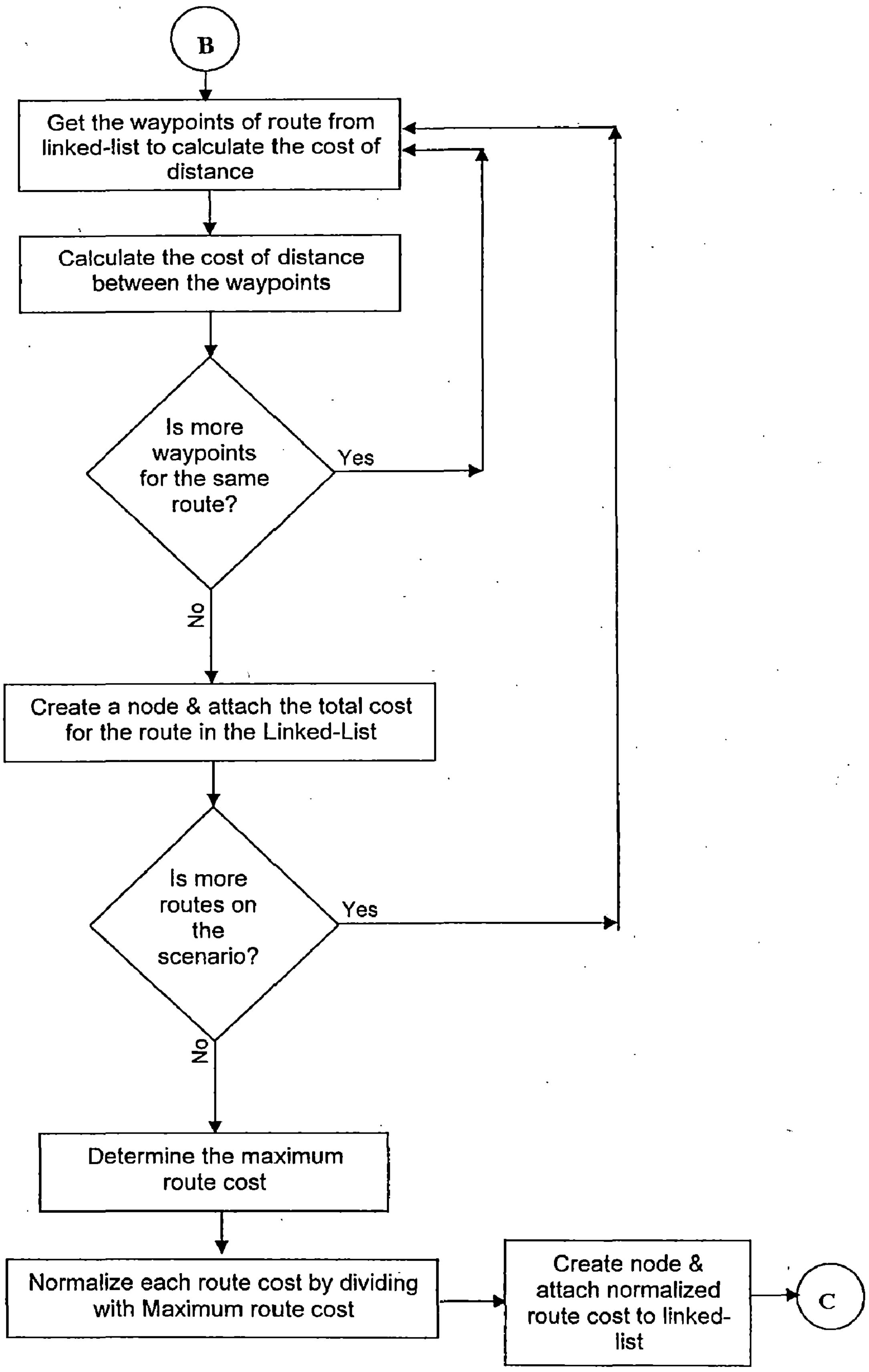
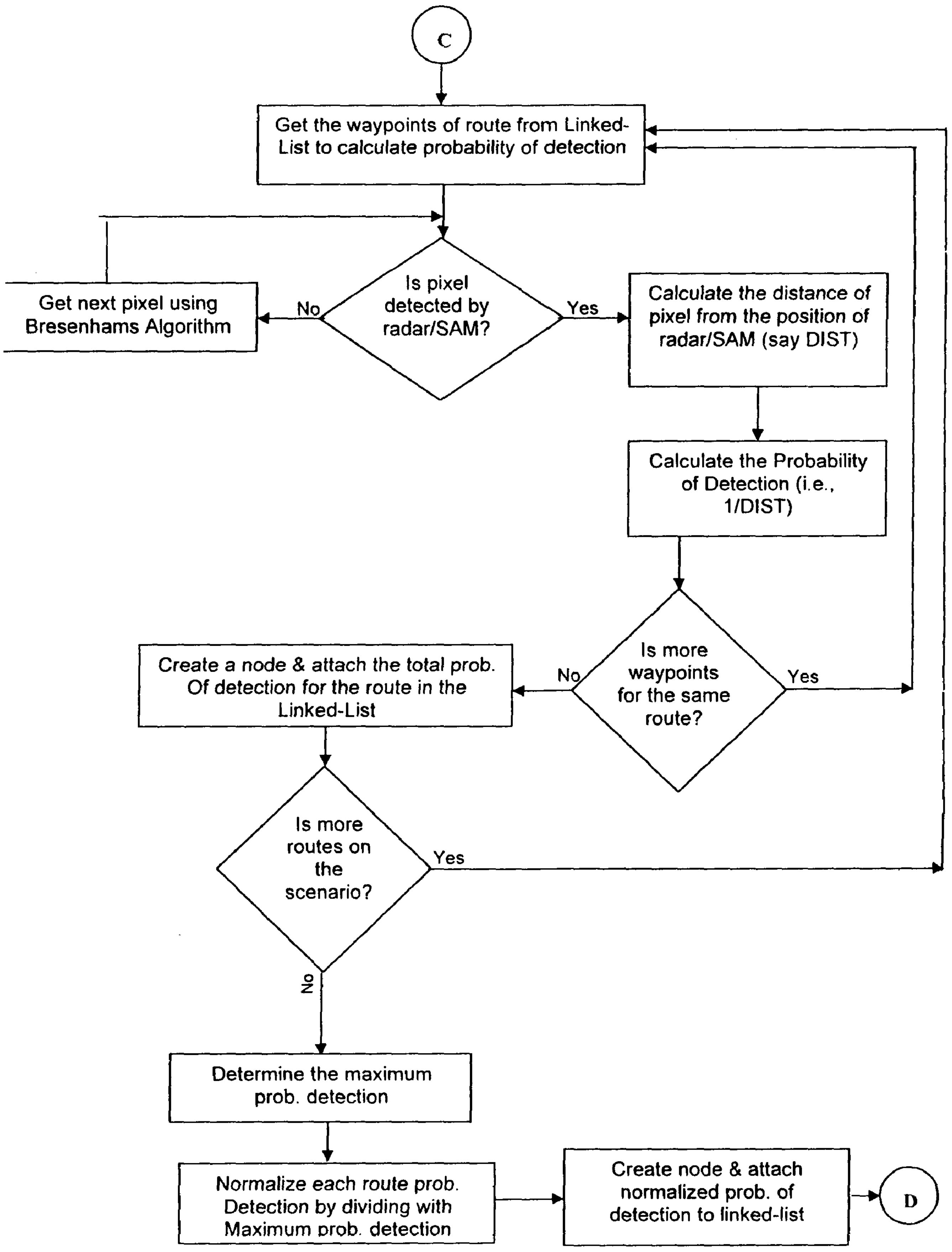

B
Get the waypoints of route from linked-list to calculate the cost of distance
Calculate the cost of distance between the waypoints
Is more waypoints for the same route?
Yes
No
Create a node & attach the total cost for the route in the Linked-List
Is more routes on the scenario?
Yes
No
Determine the maximum route cost
Normalize each route cost by dividing with Maximum route cost
Create node & attach normalized route cost to linked-list
C

Figure 5.19 Route Cost Module


C
Get the waypoints of route from Linked-List to calculate probability of detection
Is pixel detected by radar/SAM?
No
Get next pixel using Bresenhams Algorithm
Yes
Calculate the distance of pixel from the position of radar/SAM (say DIST)
Calculate the Probability of Detection (i.e., 1/DIST)
Is more waypoints for the same route?
Yes
No
Create a node & attach the total prob. Of detection for the route in the Linked-List
Is more routes on the scenario?
Yes
No
Determine the maximum prob. detection
Normalize each route prob. Detection by dividing with Maximum prob. detection
Create node & attach normalized prob. of detection to linked-list
D

G 11011

Figure 5.20 Probability Detection Module

5.3 Class Cards

Class Name: CGeographicCoord

Variable Name

Data Type/Class Type	***Name of the Variable/Object***
int	Degree
int	Minute
float	Second
char	Direction

Function name

Type of the Function	***Name of the Function***
constructor	CGeographicCoord( )
destructor	CGeographicCoord( )

Description of this class

This class basically initializes the co-ordinate position in the form of degree, minute, second and direction.

Class Name: CLoc

Variable Name

Data Type/Class Type	***Name of the Variable/Object***
CPoint	Point
CGeographicCoord	Latitude
CGeographicCoord	Longitude

Function name

Type of the Function	***Name of the Function***
CPoint	GetLoc( )
void	SetLoc( )
char*	Get_Latitude_Str( )
char*	Get_Latitude_Str( )

Description of this class

This class sets and gets the x,y co-ordinates of object as well as latitude.

Class Name: CTarget

Variable Name

Data Type/Class Type	***Name of the Variable/Object***
CLoc	Loc
Bool	is_set

Function name

Type of the Function	*Name of the Function*
constructor	CTarget( )
void	DrawTarget( )
void	SaveTarget( )
destructor	CTarget( )

Description of this class

This class is used to set, draw the target (i.e. enemy ammunitions factory) and also to save it to a file.

<u>Class Name: CWaypt</u>

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
CWaypt*	Next
CLoc	Loc
int	waypt_number
char*	route_name
float	Height
float	Speed
CTime	Time
static float	x, y
static int	deltaY, deltaX
static int	xs,ys,xf,yf
static int	DIRflag
static int	countPOINT
static int	reverseCountPOINT
static int	stopFlag

Function name

Type of the Function	*Name of the Function*
constructor	CWaypt( )
void	SetRouteName()
char*	GetRouteName( )
void	addnode( )
CWaypt*	getnode( )
virtual void	delnode( )
void	moveFighter( )
void	savelist( )
char*	ToString( )
virtual void	display( )
void	SetSpeed( )
float	GetSpeed( )
void	SetHeight( )

float	GetHeight( )
CTime	GetTime( )
void	SetTime( )
destructor	CWaypt( )

Description of this class

This class sets the location of the waypoints of the route, the height and speeds of the fighter, provides facility to specify event time in the scenario and saves all the data on Flat File using Linked list concept as well as load all the information from flat file. This class also takes care about editing and moving fighter concept.

Class Name: CRoute

Variable Name

Data Type/Class Type	***Name of the Variable/Object***
CWaypt*	waypoints
char*	route_name
float	cost_of_distance
float	normal_cost
float	prediction_cost
CRoute*	next
int	count

Function name

Type of the Function	***Name of the Function***
constructor	CRoute( )
void	SetRouteName( )
char*	GetRouteName( )
CRoute*	getnode( )
void	add_new_route( )
void	routeMovement( )
void	display( )
void	savelist( )
char*	ToString( )
CRoute*	get_route_by_name( )
Void	AddWayPt( )
Void	DeleteWaypt( )
Destructor	CRoute( )

Description of this class

This class creates the route for the Blue Force (friendly) fighter, displays it and saves it to the scenario file. It also contains the module for fighter movement.

Class Name: CGraphObject

Variable Name

Data Type/Class Type	***Name of the Variable/Object***
char*	name
Float	info
CGraphObject*	next
CLoc	Loc
char*	Type

Function name

Type of the Function	**Name of the Function**
Constructor	CGraphObject( )
Void	addnode( )
CGraphObject*	getnode( )
virtual void	delnode( )
char*	Search_node( )
Bool	Search_node( )
Void	delnode( )
char*	Search_node_by_pos( )
CRoute*	Get_route_by_name( )
CPoint	Search_node_pos( )
Void	SetType( )
char*	GetType( )
virtual void	display( )
Destructor	CGraphObject( )

Description of this class

This forms the base class for the enemy objects such as Radars, Surface-to-Air missiles and Fighters containing the features that are common to all of them i.e. location, type of the object and its name.

Class Name: CSensor

Function name

Type of the Function	***Name of the Function***
Constructor	CSensor( )
virtual void	display( )
Destructor	CSensor( )

Description of this class

This class is a base class for CRadar class.

Class Name: CRadar

Variable Name

Data Type/Class Type	Name of the Variable/Object
float	range
int	ID
float	Detect_Radius
float	Efectvness
Bool	Active
int	certain

Function name

Type of the Function	Name of the Function
constructor	CRadar( )
virtual void	display( )
CRadar*	getnode( )
void	addnode( )
virtual void	savelist( )
virtual char*	ToString( )
void	Set_Detect_Radius( )
float	Get_Detect_Radius( )
void	Set_Efectvness( )
float	Get_Efectvness( )
void	Set_Active( )
void	Set_Passive( )
void	update( )
destructor	CRadar( )

Description of this class

This class encapsulates the entire attributes specific to radars and describes their functionality related to the application. It also provides functions for addition of its objects to a linked list thereby providing dynamicity, updation of the attributes of radars during execution, drawing the objects on to the screen and also to save the information about them in the scenario file which can opened to load the scenario when required.

Class Name: CFighter

Variable Name

Data Type/Class Type	Name of the Variable/Object
float	Speed
float	Fuel_level
CRgn	Region
float	Height

Function name

Type of the Function	*Name of the Function*
Constructor	CFighter( )
void	display( )
void	addnode( )
CFighter*	getnode( )
void	savelist( )
char*	ToString( )
void	uncertdisplay( )
void	Set_Speed( )
float	Get_Speed( )
void	Set_Fuel_Level( )
float	Get_Fuel_Level( )
void	Set_Region( )
CRgn	Get_Region ( )
void	Set_Height ( )
float	Get_Height( )
void	update( )
destructor	CFighter( )

Description of this class

This class encapsulates the fighter specific features such as their height, speed, fuel level. It provides functions for adding it to the list of fighters, setting and getting its attributes, modification or deletion of their attributes during execution, drawing the objects on to the screen and also to save the information about them in the scenario file which can be opened to load the scenario when required.

Class Name: CSam

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	Detect_Radius
float	Hit_Prob

Function name

Type of the Function	*Name of the Function*
Constructor	CSam( )
char*	ToString( )
void	savelist( )
void	Set_Detect_Radius( )
float	Get_Detect_Radius( )
void	Set_Hit_Prob( )
float	Get_Hit_prob( )
void	display( )
void	addnode( )

CSam*	getnode( )
void	update( )
destructor	CSam( )

Description of this class

This class encapsulates the features specific to SAMs such as their hit probability and range. It also provides functions for adding it to the list of SAMs, setting and getting its attributes, modification or deletion of their attributes during execution, drawing the objects on to the screen and also to save the information about them in the scenario file which can be opened to load the scenario when required.

Class Name: CFighterDialogbox

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
Float	m_fuel_level
Float	m_height
Float	m_speed
CString	m_name

Function name

Type of the Function	*Name of the Function*
constructor	CFighterDialogbox(CWnd * pParent)
destructor	~CFighterDialogbox()

Description of this class

The function of this class is to provide an interface to the user to specify the parameters of the fighter such as its speed, which can vary according to the situation.

Class Name: CFighterEdit

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_fuel_level_edit
float	m_height_edit
float	m_speed_edit
CString	m_name_edit
int	m_x_value
int	m_y_value

Function name

Type of the Function	*Name of the Function*
constructor	CFighterEdit(CWnd * pParent)
destructor	~CFighterEdit ()

Description of this class

The purpose of this class is to allow the user to modify the values of the parameters of the fighter that have been set during the scenario creation time.

Class Name: CHorzDialogBar

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
int	m_x_pos
int	m_y_pos

Function name

Type of the Function	*Name of the Function*
constructor	CHorzDialogBar()
void	On ClearCanvas()
void	OnRunSim()
destructor	~ CHorzDialogBar()

Description of this class

This class specifies the buttons that have been created in order to provide a better user interface and the functions that are invoked on their selection to perform the related task.

Class Name: CRadarDialogbox

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_Detect_Radius, m_Efectvness
int	m_Active
CString	m_name

Function name

Type of the Function	*Name of the Function*
constructor	CRadarDialogBar()
destructor	~ CRadarDialogBar()

Description of this class

The function of this class is to provide an interface to the user to specify the parameters of the Radar such as its name, range, and effectiveness at the time of scenario creation.

Class Name: CVertDialogBar

Variable Name

Data Type/Class Type	Name of the Variable/Object
UINT	New_Command,select_type
int	grid_length, i
static int	Flag
CBitmap	b8,b9,b10,b12
Bool	grid_on
HBITMAP	hb1,hb2,hb3,hb4,hb5,hb6,hb7,hb8,hb9,hb10,hb11,hb12,hb13
HICON	hIcon_enemy_target,hIcon_feiendly_fighter,hIcon_enemy_fighter_certain,hIcon_enemy_fighter_uncertain,hIcon_radar_certain,hIcon_radar_uncertain,hIcon_timer,hIcon_weather,hIcon_goal,hIcon_sam_certain,hIcon_sam_uncertain,hIcon_arrow

Function name

Type of the Function	Name of the Function
constructor	CVertDialogBar()
void	SetSelectionPanelActive()
void	SetSelectionPanelInactive()
void	OnCertainPlane()
void	OnUncertainPlane()
void	OnCertainRadar()
void	OnCertainSam()
void	OnUncertainSam()
void	On Grid()
void	OnCreaeRoute()
void	OnArrow()
void	OnGoal()
void	OnSetTime()
void	OnWeather()
void	OnTargct()
void	OnFriendlyFighter()
void	OnEndRoute()
void	OnSelectRadar()
void	OnSelectSam()
void	OnSelectFighter()
void	OnSelectRoute()
void	On100Km()
void	On50Km()
void	OnEditUnit()
void	OnDeleteUnit()
destructor	~ CVertDialogBar()

Description of this class

This class specifies the buttons that have been created in order to provide a better user interface and the functions that are invoked on their selection to perform the related task.

Class Name: CWaypoint

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_Speed, m_Height
CTime	m_time

Function name

Type of the Function	*Name of the Function*
constructor	CWaypoint(CWnd* pParent)
destructor	~ CWaypoint ()

Description of this class

This class is used to specify the parameters at each waypoint.

Class Name: CScenarioAreaDlgbox

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_second_latitude,m_second_longitude
CTime	m_mission_time,
int	m_degree_latitude,m_degree_longitude,m _minute_latitude,m_minute_longitude
COLORREF	friendColor,enemyColor
UINT	m_x_extents, m_y_extents
CString	m_direction_latitude,m_direction_longitu de

Function name

Type of the Function	*Name of the Function*
constructor	CScenarioAreaDlgbox(CWnd* pParent)
void	OnFriendColor()
BOOL	OnInitDialog();
void	OnFriendColorrefPaint()
void	OnEnemyColorrefPaint()
void	OnEnemyColor()
destructor	~CScenarioAreaDlgbox()

Description of this class

This class is related to the dialog box that appears in the beginning that specifies the extent of the client area, latitude and longitude, provides facility to change the default colors of the forces involved in the air interdiction mission.

Class Name: CRadarEdit

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_detect_radius_edit,m_efectvness_edit
int	m_x_edit, m_x_edit
CString	m_name_edit

Function name

Type of the Function	*Name of the Function*
constructor	CRadarEdit(CWnd* pParent)
destructor	~CRadarEdit()

Description of this class

The purpose of this class is to allow the user to modify the values of the parameters of the radars that have been set during the scenario creation time thereby making the application flexible.

Class Name: CSamEdit

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_edit_radius, m_edit_hit_prob
int	m_x_value, m_y_value
CString	m_name_edit

Function name

Type of the Function	*Name of the Function*
constructor	CSamEdit(CWnd* pParent)
destructor	~CSamEdit()

Description of this class

The purpose of this class is to allow the user to modify the values of the parameters of the SAMs that have been set during the scenario creation time thereby making the application flexible.

Class Name: CsamDialogbox

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
float	m_detect_radius, m_hit_prob
CString	m_name

Function name

Type of the Function	*Name of the Function*
constructor	CSamDialogbox (CWnd* pParent)
void	OnReset()
destructor	~CSamDialogbox ()

Description of this class

The function of this class is to provide an interface to the user to specify the parameters of the SAM such as its name, range, and effectiveness at the time of scenario creation.

Class Name: CMydlgbarDoc

Variable Name

Data Type/Class Type	*Name of the Variable/Object*
HBRUSH	RADAR_BRUSH,SAM_BRUSH
HPEN	hDashPen,hSolidPen
COLORREF	ENEMY_COLOR, FRIEND_COLOR
Int	x_extents, y_extents
CRoute	* route1
CTarget	* target
HICON	hIcon_enemy_target,hIcon_feiendly_fighter,hIcon_enemy_fighter_certain,hIcon_enemy_fighter_uncertain,hIcon_radar_certain,hIcon_radar_uncertain,hIcon_sam_certain,hIcon_sam_uncertain
CGraphObject	* GraphObject
CRadar	* Radars
CFighter	* Fighters
CSam	* Sams

Function Name

Type of the Function	*Name of the Function*
constructor	CMydlgbarDoc()
void	Draw_All_Items(CDC* pDC)
void	SaveGlobalInfo(const char *filename)
virtual BOOL	OnNewDocument()
virtual BOOL	OnSaveDocument(LPCTSTR lpszPathName)
Virtual void	Serialize(CArchive& ar)
virtual BOOL	OnOpenDocument(LPCTSTR lpszPathName)
Virtual destructor	~CMydlgbarDoc()

Class Name: CMydlgbarView

Variable Name

Data Type/Class Type	Name of the Variable/Object
UINT	uid
int	x_extents, y_extents
CPoint	s_point, pt
float	speed,fuel_level,height
Bool	valid
HCURSOR	hCursor_my_arrow,hCursor_my_cross
HICON	fighter1,fighter2,fighter3,fighter4,fighter5,fighter6,fighter7, fighter8
LOGFONT	CurrentFont
COLORREF	ENEMY_COLOR
HFONT	hFont
CString	param1

Function name

Type of the Function	Name of the Function
constructor	CMydlgbarView()
void	run_Simulation()
CMydlgbarDoc	* GetDocument()
void	OnAlphaCall()
void	InitParams()
void	Create_New_Route()
void	set_normal_cost()
void	detection_cost()
CMydlgbar void	normalized_detect_cost()
CPoint	GetLogicalPoint(CPoint point)
float	calculate_cost();
void	OnTimer(UINT nIDEvent)
BOOL	OnEraseBkgnd(CDC* pDC)
void	OnMouseMove(UINT nFlags, CPoint point)
void	OnLButtonDown(UINT nFlags, CPoint point)
Virtual void	OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
Virtual void	OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
virtual BOOL	OnPreparePrinting(CPrintInfo* pInfo)
Virtual void	OnInitialUpdate()
virtual BOOL	PreCreateWindow(CREATESTRUCT& cs)
Virtual void	void OnDraw(CDC* pDC)
UINT	getData()
float	Get_range(float a,float b)
CPoint	Generate_random_pos(float a,float b)
destructor	~ CMydlgbarView()

CHAPTER - 6

Program Specification

6.1 Design Specifications

To accomplish the task of the interdiction mission, modeling the attributes and complex dynamics of various interacting elements is a critical issue. In order to achieve this, the attributes and behavior of each agent has been encapsulated in a structure called a class thereby following an object-oriented approach. This seems a simple task where the dynamics of the interacting elements remain the same throughout the application. But, complexity arises where there is uncertainty involved in the process. For the objects in our air force mission-planning domain, we can categorize the uncertainty into several types:

1. *Uncertainty of existence* : The object may or may not exist.
2. *Uncertainty of location* : An area of uncertainty of the object's location is available but it is not certain of the exact location of the object.
3. *Uncertainty of range* : The exact detection range or firing range is not known.
4. *Uncertainty of number* : The number of interacting objects is user specific.

Since the deployment of agents in the mission is user specific, the data structure that best suits the development of our application is a Linked list as it allow creation of objects during execution of the application thereby providing flexibility and dynamically.

6.1.1 Data Structure Used

A critical issue to be dealt with during the development of an application is the data structure used since it largely affects the performance of the application. Keeping in view of the uncertainty defined in terms of the number, range, location and existence of objects a linked list appears to be the optimal choice.

Linked List

A linked list is a dynamic data structure. Each item in the list is called a *node* and contains a minimum of two fields and an address field. The information field holds the actual data on the list while the address field contains the address of the next node in the list. The entire list is accessed from an external pointer that points to (contains the address of) the first node in the list (an "external pointer" means one that isn't included within a node rather its value can be accessed directly by referencing a variable). The next address field of the last node in the list contains a special value known as *NULL*, which isn't a valid address. This NULL pointer is used to signal the end of a list.

Advantage of Linked-List over array implementation

Under array implementation, a fixed set of nodes represented by an array is established at the start of the execution. A pointer a node is represented by the relative position of the node within the array. The disadvantage of this approach is two fold:

- The number of nodes that are needed often cannot be predicted when a program is written. Usually, the data with which the program is executed determines the number of nodes necessary. Thus no matter how many elements of the array of nodes contains, it is always possible that the program will be executed with input that requires a larger number.
- Whatever numbers of nodes are declared must remains allocated to the program throughout its execution?

The solution to this problem is to allow nodes that are dynamic rather than static i.e., when a node is needed, storage is reserved for it and when it is no longer needed the storage is released. The storage for nodes that are no longer in use is available for another purpose. Also, no predefined limit on the number of nodes is established. As long as sufficient storage is available to the job as a whole, part of that storage can be reserved for use as a node.

6.1.2 Scenario File Specification

The idea behind maintaining a scenario file is to provide the user with the facility to restore the scenario so long as the deployment of the enemy objects that are certain remains fixed during the simulation process. The file not only contains the global parameters of the scenario such as the extents of the client area, the complete description of various enemy units including their name, type, location, and other information specific to each of them, but also the complete details of the routes to be simulated. This file serves as an input for the simulation module.

Scenario File Format :

```
        # Global parameters #
        $ radar1 : attrib1 : attrib2 :attrib3 $
        $ radar2 : attrib1 : attrib2 :attrib3 $
        .
        .
        .
        $ fighter1 : attrib1 : attrib2 :attrib3 $
        $ fighter2 : attrib1 : attrib2 :attrib3 $
        .
        .
        .
        $ sam1 : attrib1 : attrib2 :attrib3 $
        $ sam2 : attrib1 : attrib2 :attrib3 $
        .
        .
        .
        $ route1 : waypt1 : waypt2 : waypt3 $
        $ route2 : waypt1 : waypt2 : waypt3 $
        .
        .
        .
```

The # symbol is used to mark the header and trailer of the block specifying the global parameters.

The $ symbol is used to mark the header and trailer of the records for each object whose attribute are separated by colons (:).

6.2 Implementation

This phase deals with the preparation of the Graphical User Interface (GUI) to provide the user with an environment exhibiting air interdiction mission, where the target is to destroy the enemy ammunitions factory.

We categorize the phases involved in the development of the application as:

I GUI development phase

II Simulation module

6.2.1 GUI Development

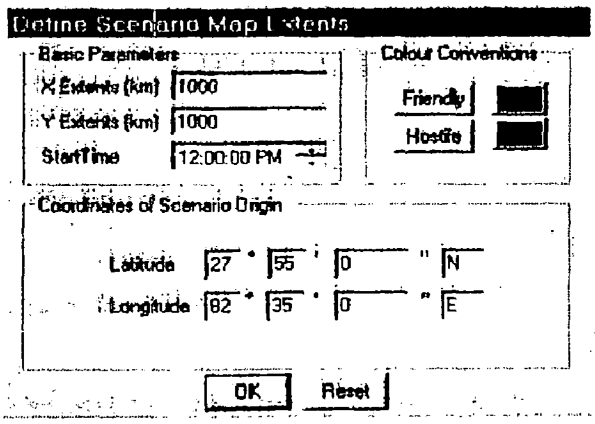
The GUI for the interdiction mission is designed taking into consideration the input output requirements of the application as well as the relationship that exists between various objects actively involved in the interdiction mission. The GUI focuses on the setting of the client area where the simulation can proceed.

Features supported by the GUI include: -

1. **Scenario Map Extents Dialog Box**

   The interface for the scenario map extents shown in Fig 6.1 specifies

   - The extents of the client area indicating the span of the mission.
   - It provides facility to the user to specify the color for the two forces involved in the mission apart from the default color i.e. red for the enemy and blue for the friendly forces.
   - Values for latitude and longitude can also be specified.


Define Scenario Map Extents

Basic Parameters

X Extents (km) 1000

Y Extents (km) 1000

StartTime 12:00:00 PM

Colour Conventions

Friendly ■

Hostile ■

Coordinates of Scenario Origin

Latitude 27 ° 55 ' 0 " N

Longitude 82 ° 35 ' 0 " E

Figure 6.1 Scenario Map Extents

2. Dialog Bars

Dialog bars derived from Control bars greatly enhance a program's usability by providing quick, one-step command actions. As in a dialog box, the user can tab among the controls. Dialog bars can be aligned to the top, bottom, left, or right side of a frame window.

***Left Dialog Bar*:**

Interface for the vertical dialog bar shown in Fig 6.2(a) includes:

- Buttons for the certain and uncertain Red force (i.e. enemy) units i.e. Radars, Surface-to-Air missiles and enemy fighters.
- Button that depict the Target i.e. Red Force ammunitions Factory.
- Buttons that invoke the functionality for route creation.
- Button that shows the Blue Force (friendly) fighter.
- Buttons associated with the modules for editing and deletion of objects during execution.
- Buttons to specify the goal of the interdiction mission.
- Button to record the time taken for simulation.
- Button to indicate weather condition.
- Button for the grid specification.

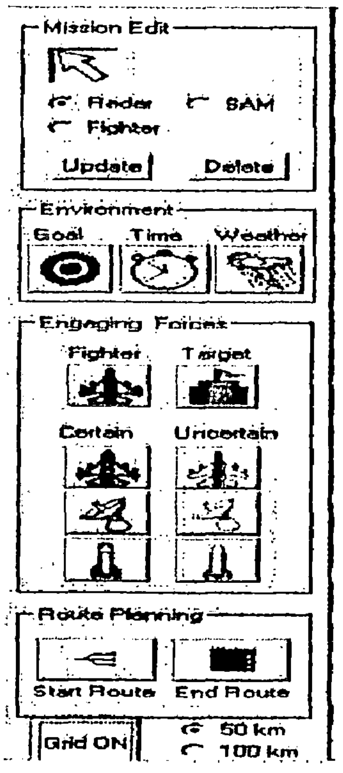


Figure 6.2(a) Left Dialog Bar

Right Dialog bar:

Right dialog bar depicted in Fig 6.2(b) serves as an interface to record the result of the simulation module. This dialog bar provides facility for:

- Recording the route name and the total score calculated for each route. This score is calculated based upon two factors – the cost of the route in terms of distance and the probability of getting detected on that route.

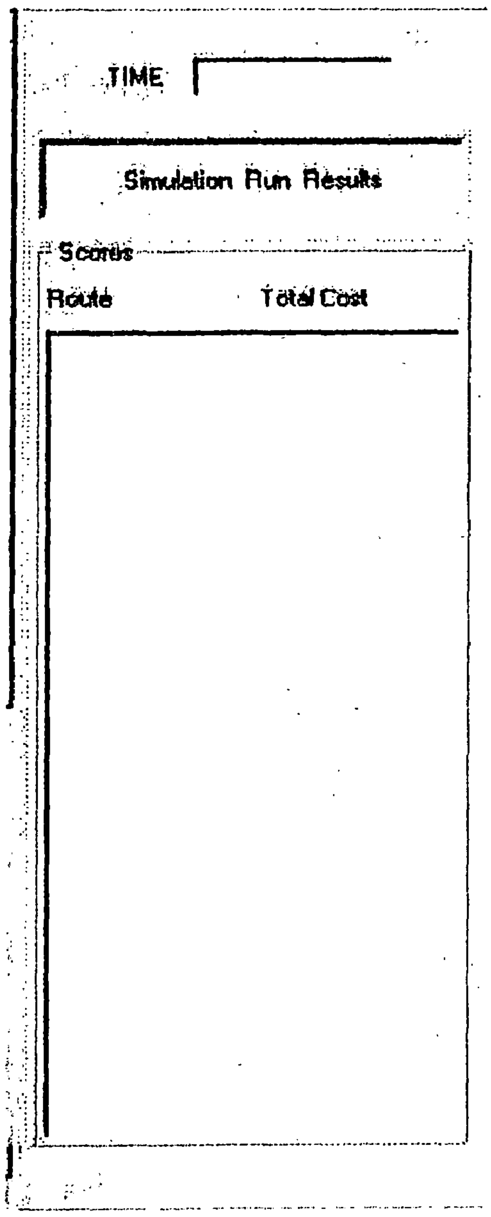
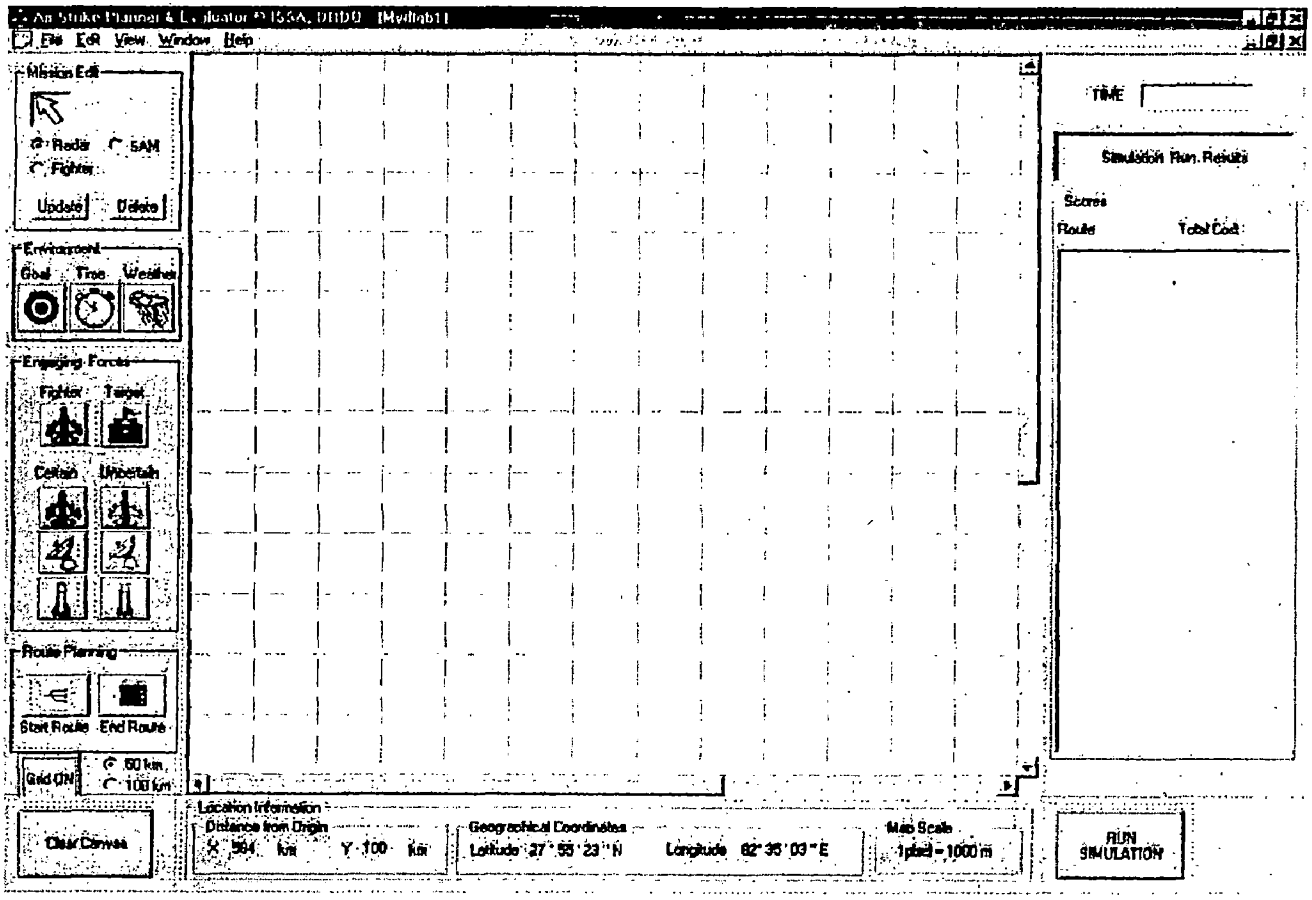


Figure 6.2(b) Right Dialog Bar



(● 50 km ○ 100 km)

Location Information
Distance from Origin: X 564 km Y 100 km
Geographical Coordinates: Latitude 27° 55' 23'' N Longitude 82° 35' 03'' E
Map Scale: 1pixel = 1000 m

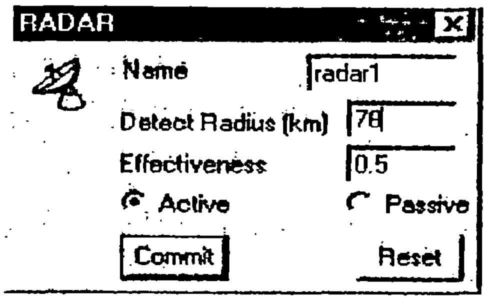
Figure 6.2(c) An Overview of Scenario

3. Interface for the initializing the attributes of Red Force units: -

a. Radar:

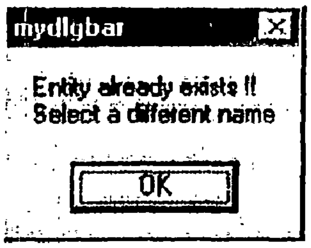
Fig 6.3 shows the interface for setting the attributes of the radars that donot includes any uncertainty to be modeled. The attributes of the radar include:

- Name of the radar
- Range of the radar
- Effectiveness
- Active/Passive to indicate whether its active or passive



Name: radar1
Detect Radius (km): 78
Effectiveness: 0.5
[✓ Active] Passive

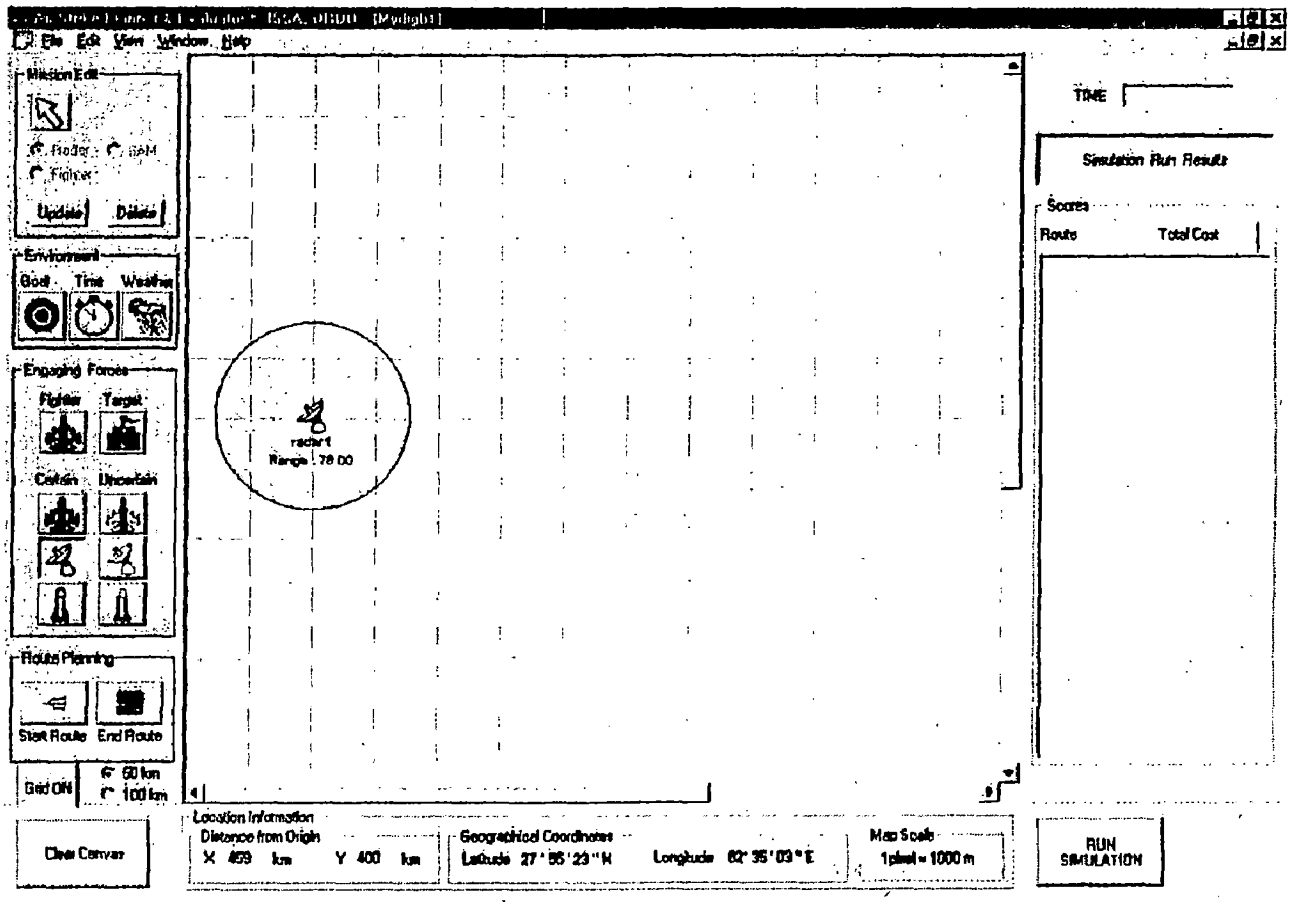
Figure 6.3 Parameters for Hostile Radar


mydlgbar

Entity already exists !!
Select a different name

OK

Figure 6.4 Check-Box for duplicity in name



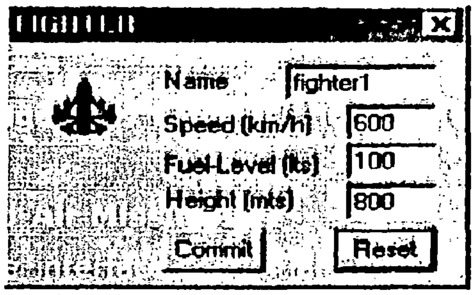
Location Information
Distance from Origin X 459 km Y 400 km
Geographical Coordinates Latitude 27 ' 56 ' 23 " N Longitude 62° 35 ' 03 " E
Map Scale 1pixel = 1000 m

Figure 6.5 Hostile Radar Creation

b. Fighter:

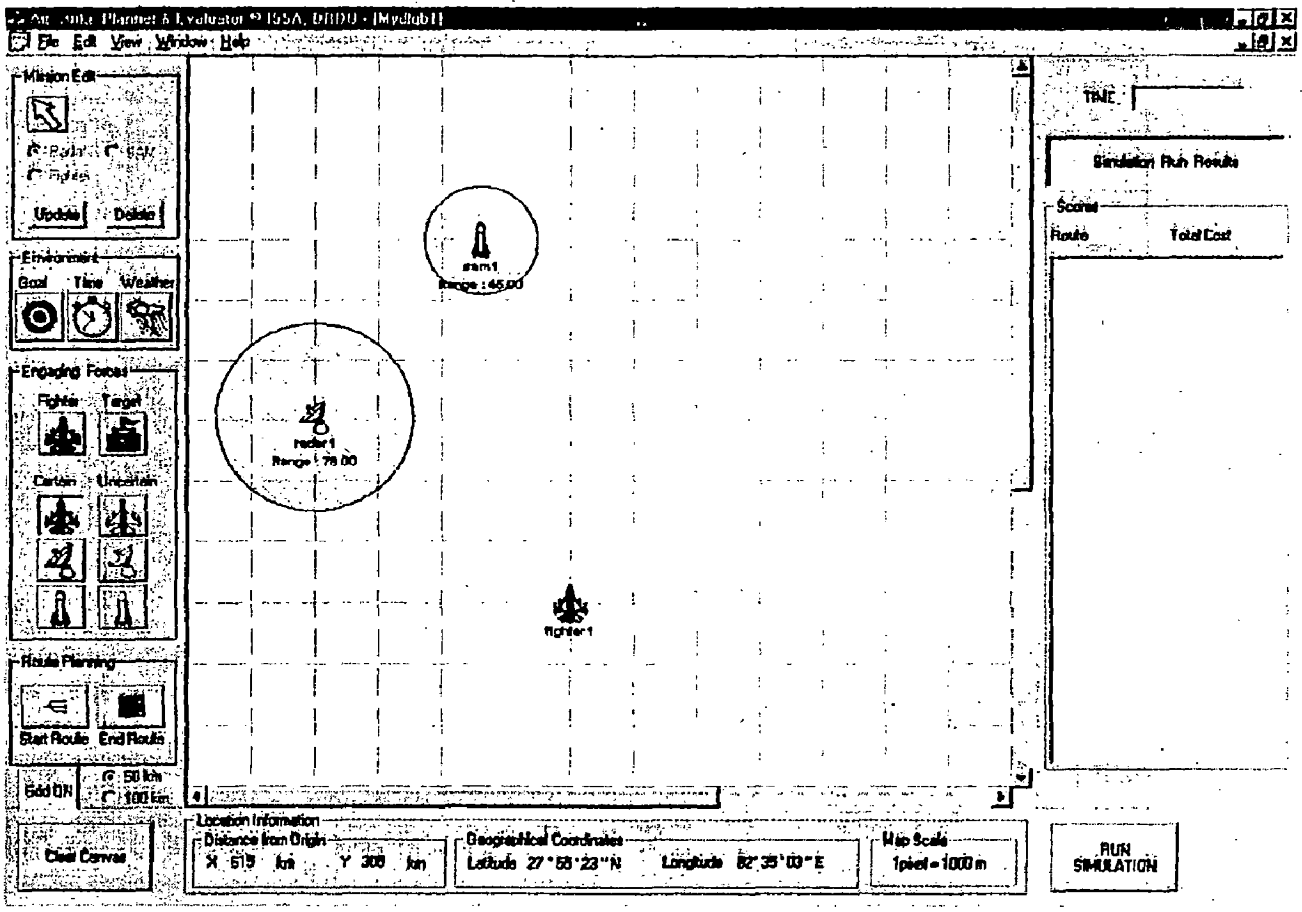
Fig 6.6 shows the interface for initializing the attributes of the Fighters that don't involve any uncertainty to be modeled. The attributes of fighter include:

- Name of the fighter
- Height
- Speed of the fighter
- Fuel-level


FIGHTER

Name: fighter1
Speed (km/h): 600
Fuel-Level (lts): 100
Height (mts): 800

Figure 6.6 Parameters for Enemy Fighter



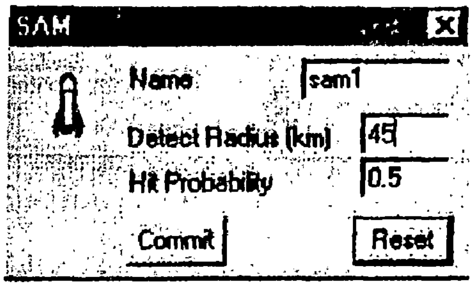
Location Information
Distance from Origin: X 619 km Y 300 km
Geographical Coordinates: Latitude 27°58'23"N Longitude 82°35'00"E
Map Scale: 1pixel = 1000 m

Figure 6.7 Enemy Fighter Creation

c. SAM (Surface-to-Air Missile):

Fig 6.8 shows the interface for initializing the attributes of the Fighters that don't involves any uncertainty to be modeled. The attributes of fighter include:

- Name of the SAM
- Range
- Hit-Probability
- Fuel-level


SAM

Name: sam1

Detect Radius (km): 45

Hit Probability: 0.5

Figure 6.8 Parameters for Hostile SAM

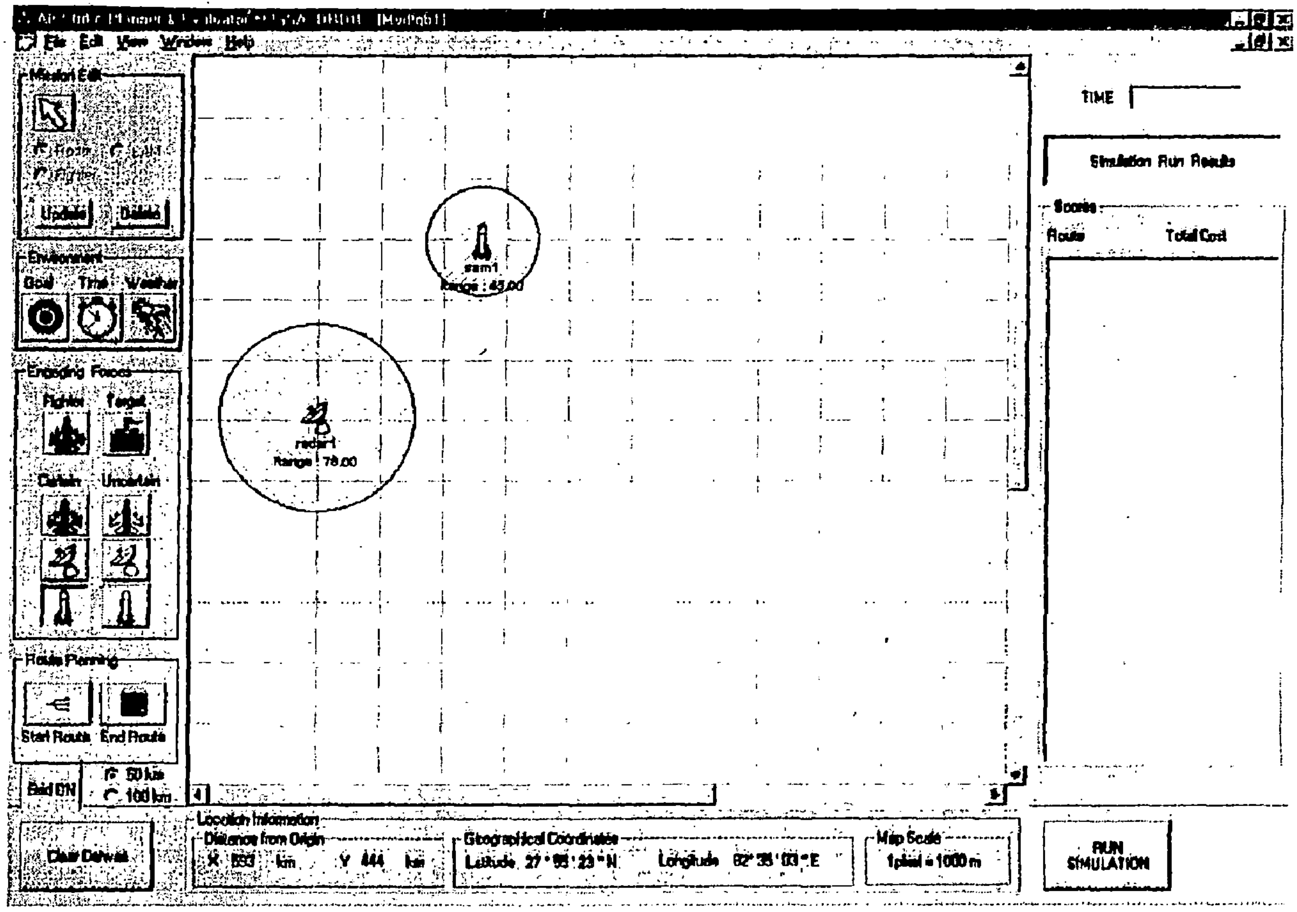
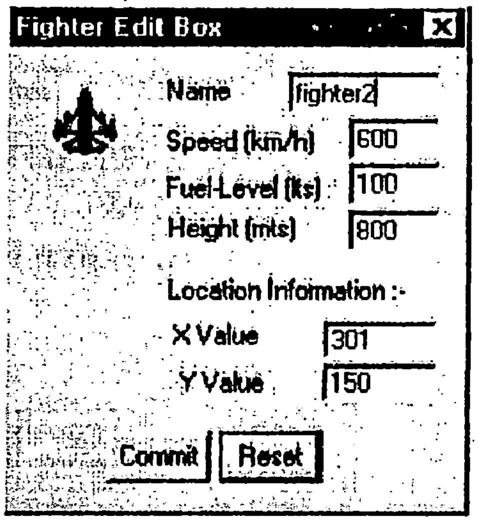


Figure 6.9 Hostile SAM Creation

4. **Interface providing editing and deleting facility:**

   Editing:

   The GUI presents the user,with the facility to edit the attributes of the units deployed including their location to modify the scenario prior to the execution of the simulation module.



Fighter Edit Box

Name fighter2

Speed (km/h) 600

Fuel-Level (lts) 100

Height (mts) 800

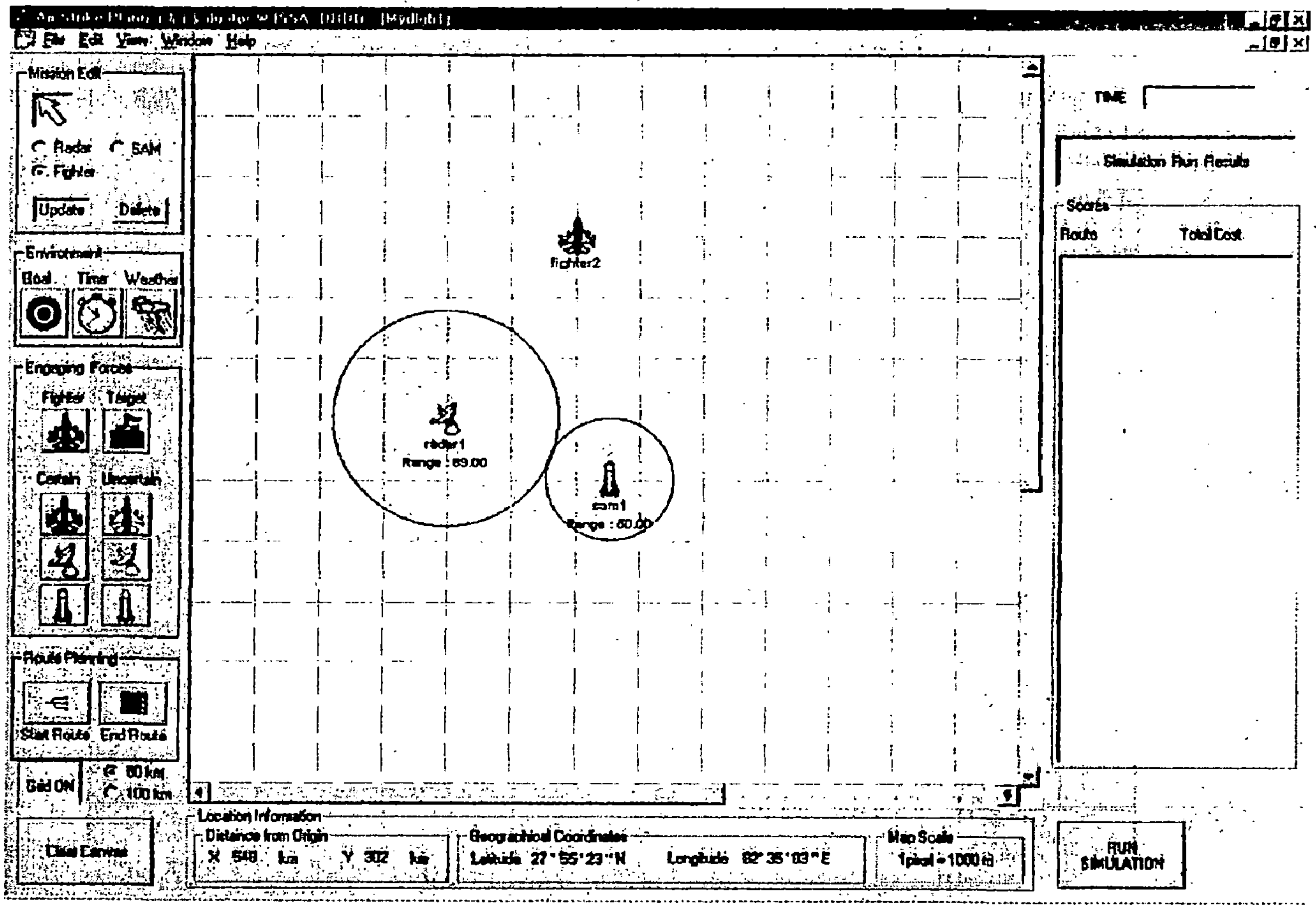
Location Infoimation :-

X Value 301

Y Value 150

Commit Reset

Figure 6.10 Fighter Edit Box



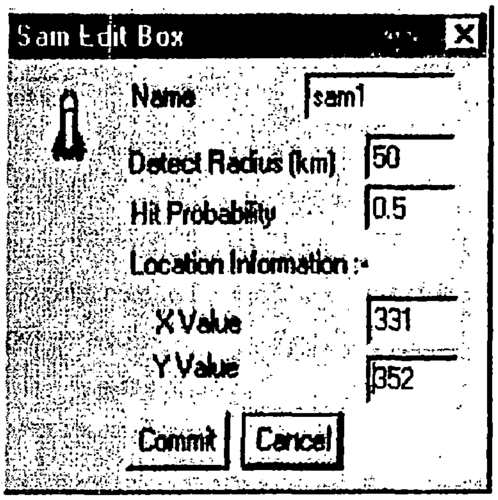
Location Information

Distance from Origin: X 548 km Y 302 km

Geographical Coordinates: Latitude 27° 55' 23" N Longitude 82° 35' 03" E

Map Scale: 1pixel = 1000 ft

Figure 6.11 Scenario after Editing Fighter



Sam Edit Box

Name: sam1

Detect Radius (km): 50

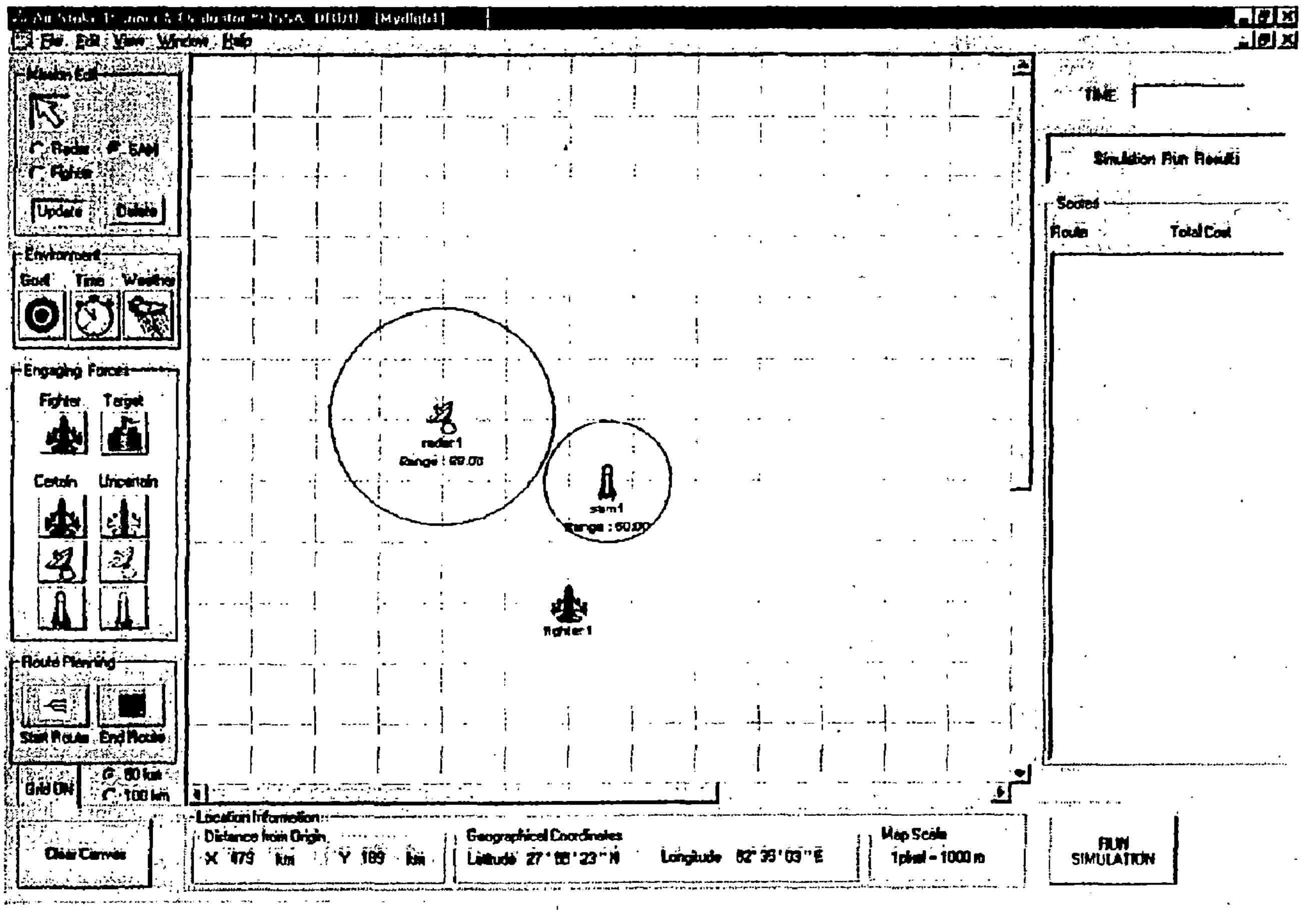
Hit Probability: 0.5

Location Information :-

X Value: 331

Y Value: 352

Figure 6.12 SAM Edit Box



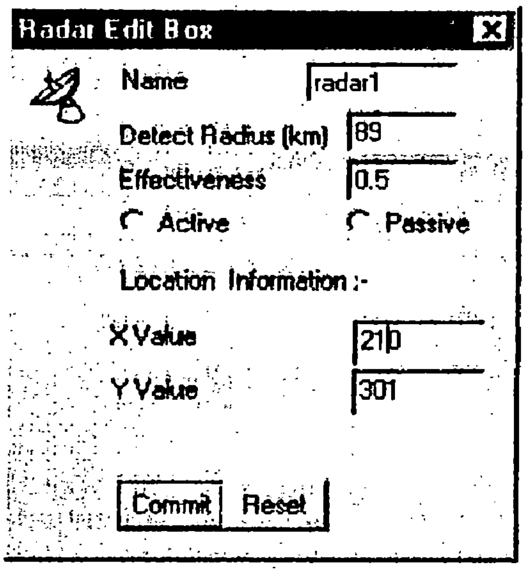
Location Information

Distance from Origin: X 479 km Y 189 km

Geographical Coordinates: Latitude 27° 18' 23" N Longitude 82° 39' 03" E

Map Scale: 1pixel = 1000 m

Figure 6.13 Scenario after Editing SAM

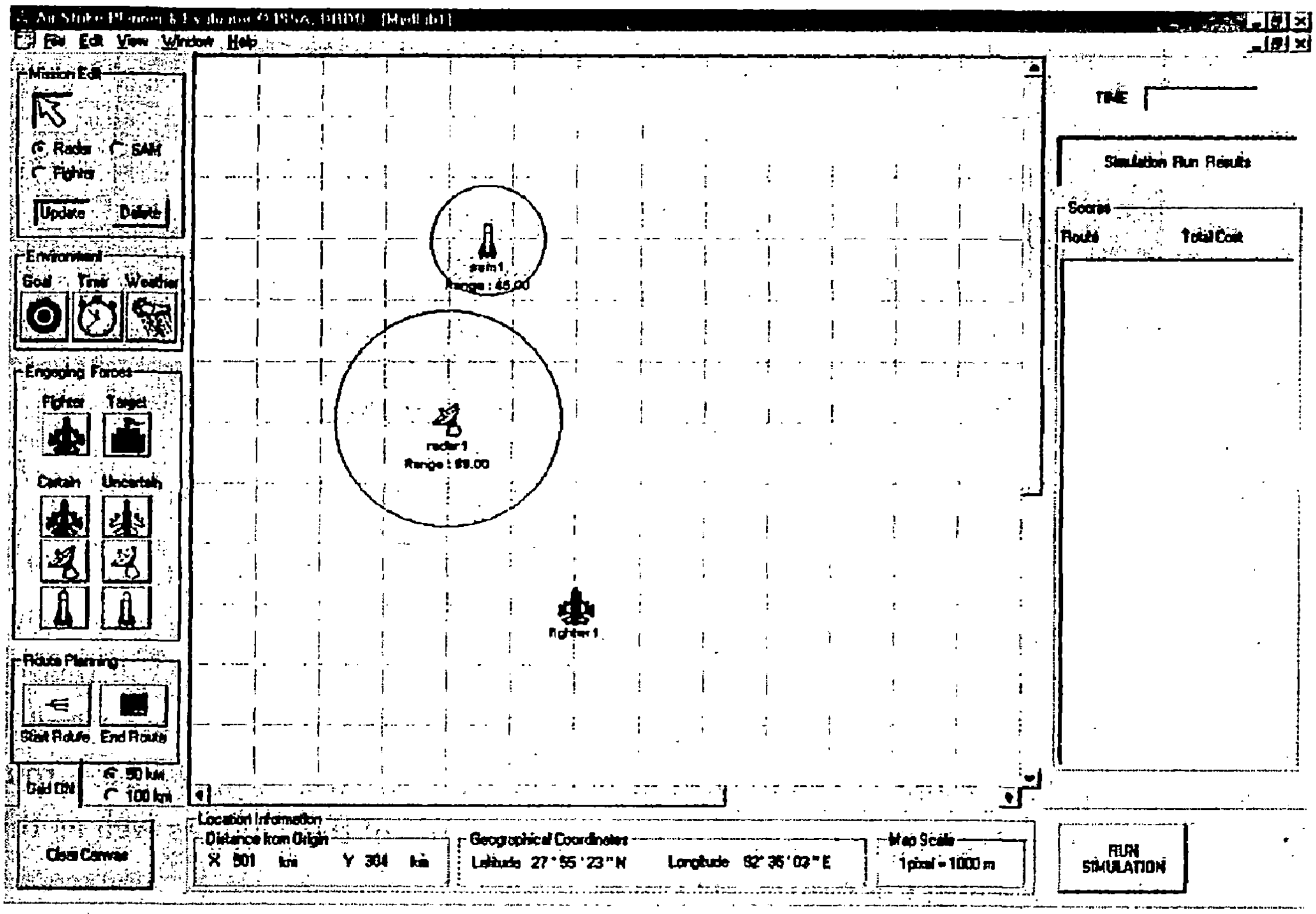


Radar Edit Box

Name: radar1
Detect Radius (km): 89
Effectiveness: 0.5
Active / Passive
Location Information :-
X Value: 210
Y Value: 301

Commit | Reset

Figure 6.14 Radar Edit Box

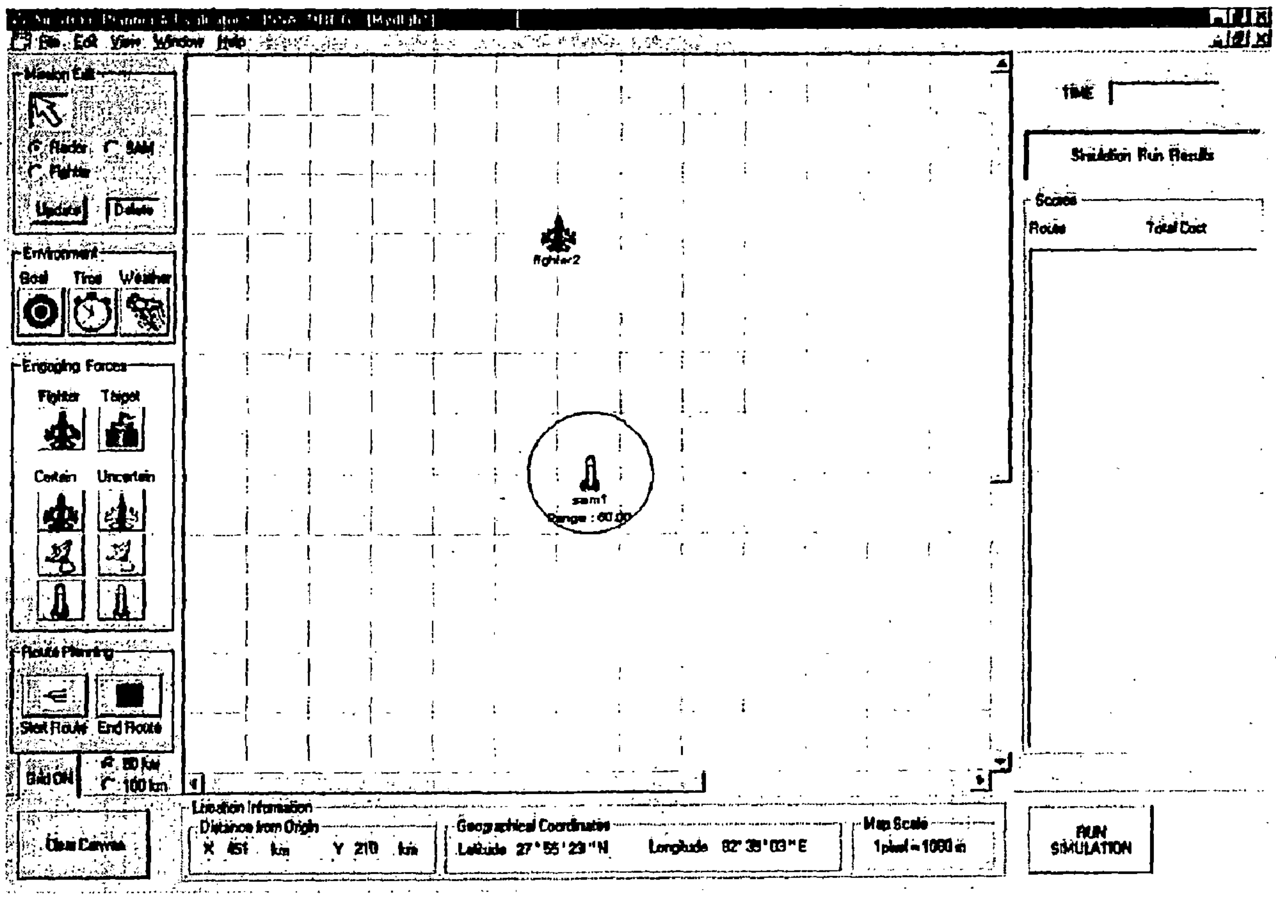


Location Information
Distance from Origin: X 501 km Y 304 km
Geographical Coordinates: Latitude 27°55'23"N Longitude 82°36'03"E
Map Scale: 1pixel = 1000 m

Figure 6.15 Scenario after Editing Radar

Deleting:

The GUI also provides support for deleting the deployed units if any one of them becomes useless (i.e. looses its significance) for the interdiction mission. This could happen if any of the units gets destroyed or losses its effectiveness.

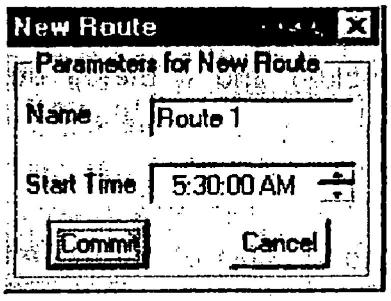


Location Information
Distance from Origin: X 451 km Y 210 km
Geographical Coordinates: Latitude 27°55'23"N Longitude 82°35'03"E
Map Scale: 1pixel = 1000 m

Figure 6.16 Scenario after Deleting Radar

5. Interface for route creation:

The aim of the interdiction mission is to guide a Blue Force (friendly fighter) to destroy Red Force (enemy) ammunitions factory to delay its supply and to come back to its base safely given a number of threats to the mission. This interface allows the user to create a number of routes that are simulated and a best route is then selected depending upon the score of each route. The interface allows the user to specify the name of the route as well as the speed with which the fighter will travel.


New Route

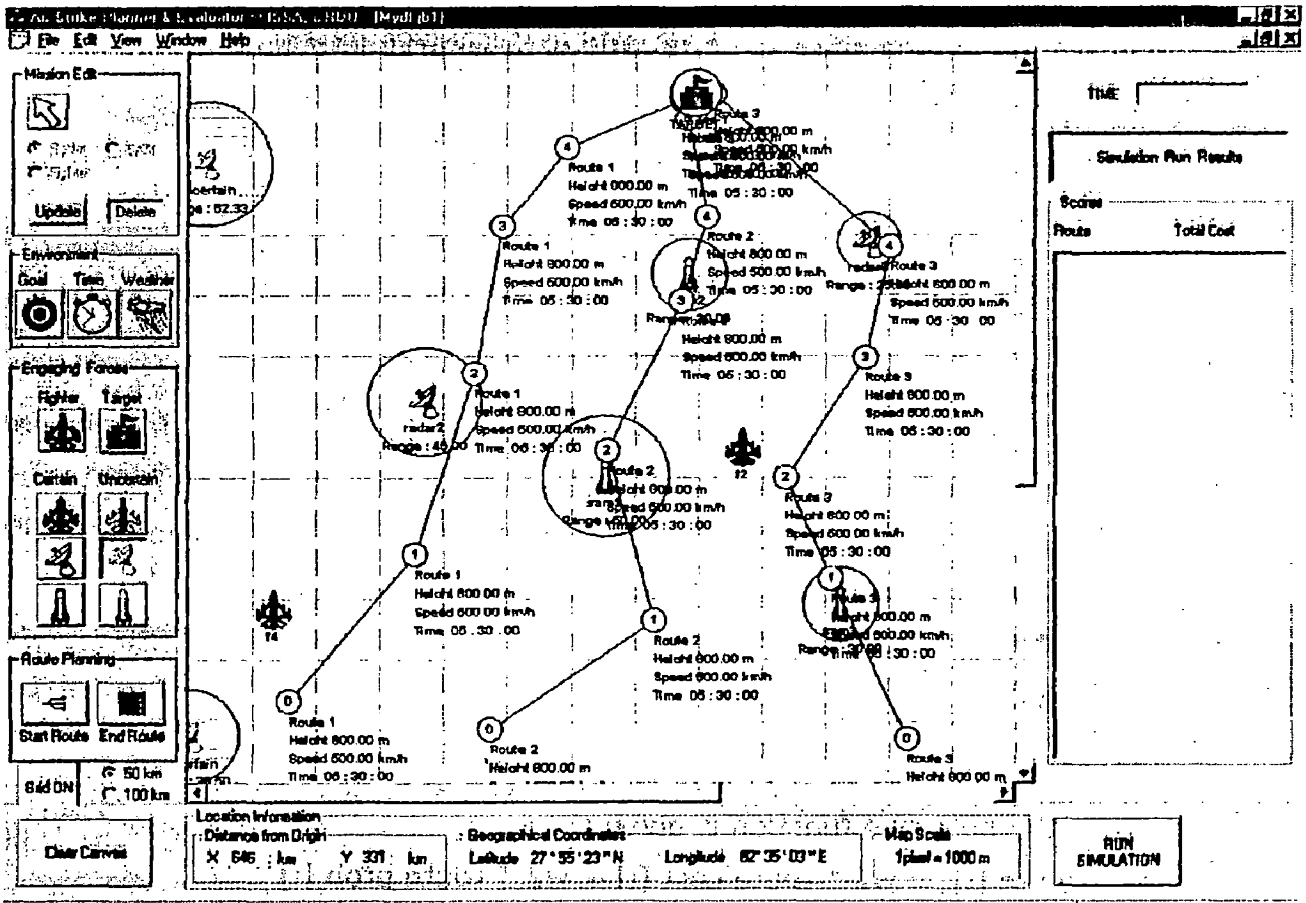
Parameters for New Route

Name: Route 1

Start Time: 5:30:00 AM

Commit Cancel

Figure 6.17 Parameters for new Route



Location Information

Distance from Origin: X 646 km Y 331 km

Geographical Coordinates: Latitude 27°55'23"N Longitude 82°35'03"E

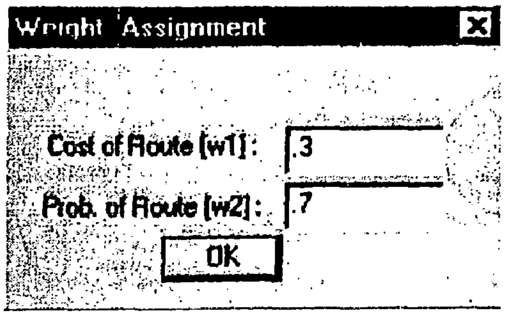
Map Scale: 1pixel = 1000 m

Figure 6.18 A scenario Created for Simulation

6.2.2 Simulation Module

Once the GUI has been created, it is then ready to be provided as an input to the simulation module. Prior to invoking the module associated with the simulation button uncertainties in terms of the existence, range etc of the objects has to be resolved. Click on the uncertain radar/SAM button to place the same on the scenario. This causes a function associated with these buttons to invoke and random number streams are used to resolve the uncertainties related to the selected objects regarding their existence, range etc. Now the function associated with the Run Simulation button is invoked and a dialog box appears asking the user to give weightage to the factors upon which the simulation module operates. These factors are: -

- The weight attached to the cost of the route in terms of distance
- The weight attached to the probability of being detected on the route.



Cost of Route [w1]: .3
Prob. of Route [w2]: .7

Figure 6.19 Weight Assignment

After specification of the weights associated with the guiding factors each route is examined to fine out the total cost of the route in terms of distance, the total probability of detection and the normalized cost and detection probability. Once all the routes have been examined calculate the total score for each route by multiplying the costs with their respective weights. Finally, the route with the minimum score is selected to be the best route, which the Blue Force fighter will follow to reach the target.

Thus the objective function is given as: -

Route [i] = w1 * route cost [i] + w2 * route prob [i]

where,

i = Route No.

Route [i] = Total cost of the route i

route cost [i] = Total cost of route i in terms of distance

route prob [i] = Total probability of being detected on route i

w1 = Weight attached to cost of route i

w2 = Weight attached to probability of detection for route i

The simulation algorithm is as follows: -

1. Assign the weights w1 and w2 to the factors guiding the simulation process.

2. For each route do

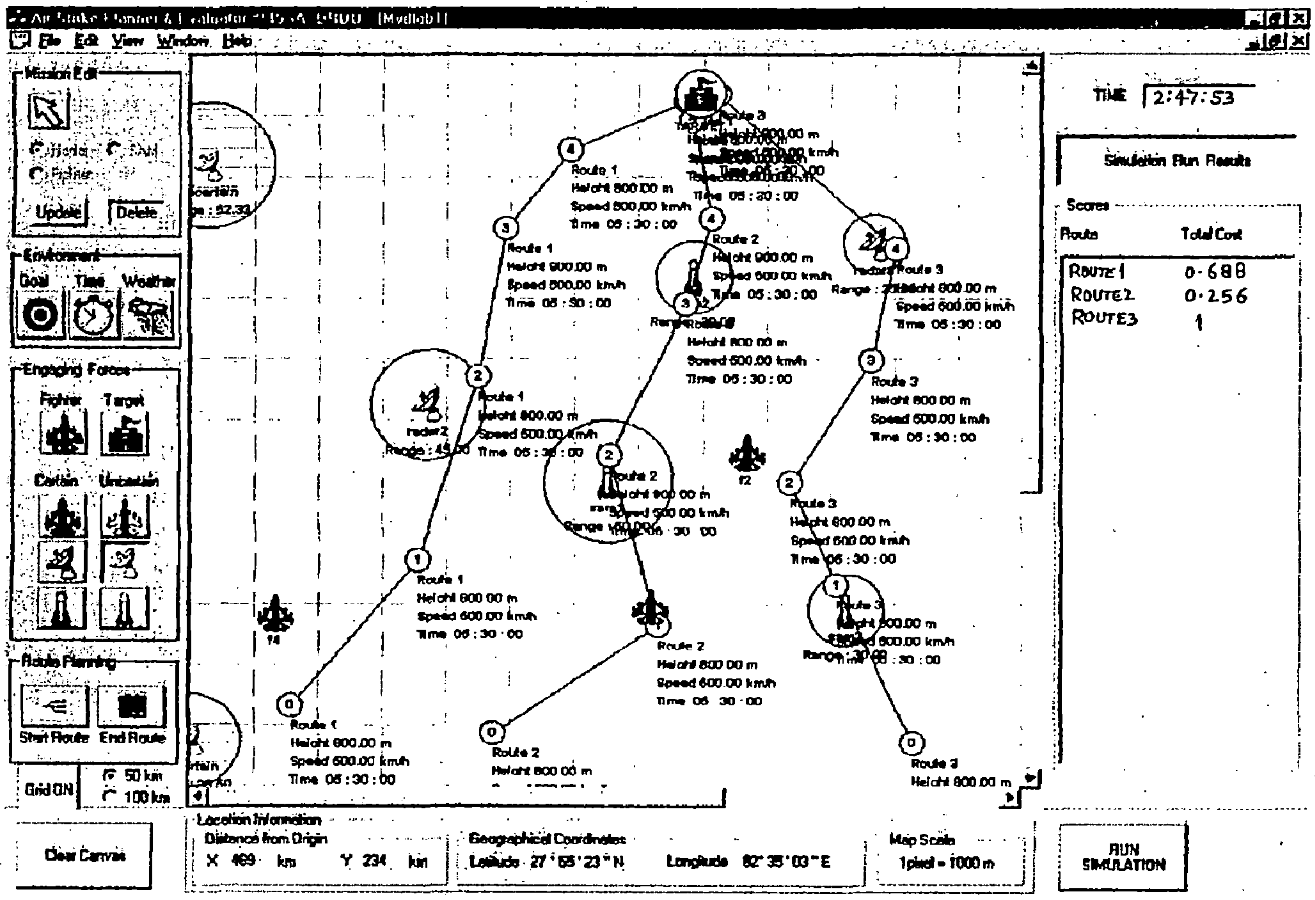
    2.1.1 calculate total cost of distance

    2.1.2 calculate the total probability of detection

    2.1.3 divide total cost of distance with the maximum cost of all the routes examined so far to find out the normal cost

    2.1.4 divide total probability of detection with the maximum probability of all the routes examined so far to find our the normal probability.

1. Using the objective function calculate the score for each route and select the path with the minimum score as the optimal path.
2. Make the fighter to follow the selected path to reach the target for its destruction.



TIME 2:47:53

Route	Total Cost
ROUTE 1	0.688
ROUTE 2	0.256
ROUTE 3	1

Location Information
Distance from Origin X 469 km Y 234 km
Geographical Coordinates Latitude 27°55'23"N Longitude 82°35'03"E
Map Scale 1pixel = 1000 m

Figure 6.20 Scenario after Simulation

CHAPTER - 7

Conclusion & Suggestions for Future Work

7.1 Conclusion

The application developed in VC++ compiles successfully and simulate the aircraft route planning for the optimal path on which an aircraft can move to achieve the target decided. The target which had to attain during this period for this application was successfully achieved.

This application has been developed for military training purpose which will be used to train the military personnel for finding optimal route from the different possible route to hit the target in the war scenario where the certain/uncertain radars and SAMs of different ranges with red fighter (enemy fighter) are ready to detect and hit the blue fighter (friendly fighter).

7.2 Suggestions for Future Work

There is no restriction for modification in the software or application which has been made.

1. Currently this application has been developed for the resolution of 1024x780, but it can be developed resolution free.
2. The method for the detection of probability can be use other than what we have used.
3. Some other factors like fuel, speed etc can also be incorporated in the simulation.

References

[1]

Fishwick,P.A., Kim,G., and Lee, J.J., SIMULATION. Nov. 1996, "**Improved Decision Making Through Simulation-Based Planning**".

[2] Law, A.M., and Kelton,W.D. 1991, "**Simulation Modelling and Analysis**" McGraw-Hill, New York.

[3] Kruglinski,D.J., 1997, "**Inside Visual C++**", 4th Edition, Microsoft Press, U.S.A.

[4] **MSDN** (Micro-Soft Developer Networks), Jan. 2002.

[5] http://www.combataircraftsimulation.php4hosting.com

[6] http://www.afrihorizons.com/index.html

[7] http://ubmail.ubalt.edu/~harsham/simulation

[8] http://www.santafe.edu

APPENDIX-A

Introduction to VC++ Architecture

Visual C++ is a powerful and complex tool for building 32-bit applications for Window 95 and Windows NT. These applications are far larger and more complex than their predecessors for 16-bit Windows, or older programs that did not use a graphical user interface. Yet as program size and complexity has grown, programmer effort has actually decreased, at least for programmers who are using the right tools.

Visual C++ is one of the right tools. With its code generating Wizards it can produce the shell of a working Windows application in seconds. The class library included with Visual C++, the Microsoft Foundation Classes, has become the industry standard for Windows software development in a variety of C++ compilers. The visual editing tools make layout of menus and dialogs a snap.

Visual C++ doesn't just compile code, it generates code. You can create a Windows application in minutes by telling AppWizard to make you a "starter app" with all the Windows boilerplate code you want. AppWizard is a very effective tool. It copies code that almost all Windows applications need into your application. An application with resizable edges, minimize and maximize buttons, a File menu with Open, Close, Print Setup, Print, and Exit options etc.. AppWizard makes skeleton, executable Windows programs in less than a minute.

Other Applications AppWizard Can Make

Other application generating wizards can make DLLs, ActiveX controls, console applications, libraries, make file, Internet Server extensions and filters, and more. Microsoft Windows was designed long before the C++ language became popular. Because thousands of applications use the C-language Windows application-programming interface (API), that interface will be maintained for the foreseeable future. Any C++ Windows interface must therefore be built on top of the procedural C-language API. This guarantees that C++ applications will be able to coexist with C applications. The Microsoft Foundation Class Library is an object-oriented interface to Windows that meets the following design goals:

- Significant reduction in the effort to write an application for Windows.
- Execution speed comparable to that of the C-language API.
- Minimum code size overhead.
- Ability to call any Windows C function directly.
- Easier conversion of existing C applications to C++.
- Ability to leverage from the existing base of C-language Windows programming experience.
- Easier use of the Windows API with C++ than with C.
- Easier-to-use yet powerful abstractions of complicated features such as ActiveX, database support, printing, toolbars, and status bars.
- True Windows API for C++ that effectively uses C++ language features.

MFC: Overview

The Microsoft Foundation Class Library (MFC) is an "application framework" for programming in Microsoft Windows. Written in C++, MFC provides much of the code necessary for managing windows, menus, and dialog boxes; performing basic input/output; storing collections of data objects; and so on. All you need to do is add your application-specific code into this framework. And, given the nature of C++ class programming, it's easy to extend or override the basic functionality the MFC framework supplies.

The MFC framework is a powerful approach that lets you build upon the work of expert programmers for Windows. MFC shortens development time; makes code more portable; provides tremendous support without reducing programming freedom and flexibility; and gives easy access to "hard to program" user-interface elements and technologies, like ActiveX, OLE, and Internet programming. Furthermore, MFC simplifies database programming through Data Access Objects (DAO) and Open Database Connectivity (ODBC), and network programming through Windows Sockets. MFC makes it easy to program features like property sheets ("tab dialogs"), print preview, and floating, customizable toolbars.

What MFC Can Do for You

The classes in MFC, taken together, constitute an "application framework". It is the framework of an application written for the Windows API. Your programming task is to fill in the code that is specific to your application. Despite its generality, MFC does support you in many specialized ways support for

- OLE visual editing.
- Automation.
- ActiveX Controls
- Internet programming.
- Windows Common Controls.
- DAO Database Programming.
- ODBC Database Programming.
- Multithreaded Programming.
- Windows Sockets for Network Programming.
- Portability

General Class Design Philosophy

MFC supplies class CWnd to encapsulate the HWND handle of a window. The CWnd

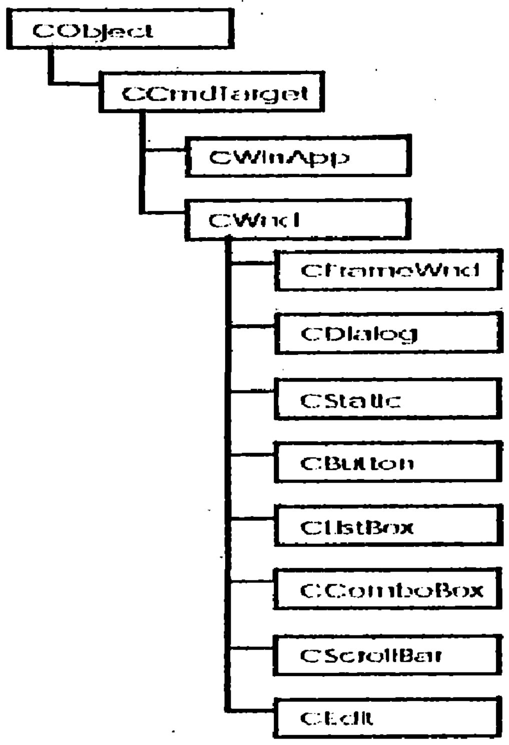


Figure a(i) The portion of the Microsoft Foundation Class Library that deals with applications and windows.

object is a C++ window object, distinct from the HWND that represents a Windows window but containing it. Use CWnd to derive your own child window classes, or use one of the many MFC classes derived from CWnd. Class CWnd is the base class for all windows, including frame windows, dialog boxes, child windows, controls, and control bars such as toolbars.

MFC uses classes CFrameWnd, CMDIFrameWnd, and CMDIChildWnd to represent single document interface (SDI) and multiple document interface (MDI) frame windows.

MFC manages windows, but you can derive your own classes and use CWnd member functions to customize these windows. You can create child windows by constructing a CWnd object and calling its Create member function, then manage the child windows with other CWnd member functions. You can embed objects derived from CView, such as form views or tree views, in a frame window. And you can support multiple views of your documents via splitter panes, supplied by class CSplitterWnd.

There are several things to notice in Figure above. First, most classes in MFC derive from a base class called CObject. This class contains data members and member functions that are common to most MFC classes. The second thing to notice is the simplicity of the diagram. The CwinApp class is used whenever you create an application and it is used only once in any program. The CWnd class collects all the common features found in windows, dialog boxes, and controls. The CFrameWnd class inherits from CWnd and implements a normal framed application window. CDialog handles the two normal flavors of dialogs: modeless and modal, respectively. Finally, Windows supports six native control types: static text, editable text, push buttons, scroll bars, lists, and combo boxes (an extended form of list). The other classes in the MFC hierarchy implement other features such as memory management, document control, database support, and so on.

Each object derived from class CWnd contains a message map, through which you can map Windows messages or command IDs to your own handler functions for them.

Messages and Commands: Overview

In traditional programs for Windows, Windows messages are handled in a large switch statement in a window procedure. MFC instead uses message maps to map direct messages to distinct class member functions. Message maps are more efficient than virtual functions for this purpose, and they allow messages to be handled by the most appropriate C++ object—application, document, view, and so on. You can map a single message or a range of messages, command IDs, or control IDs.

WM_COMMAND messages—usually generated by menus, toolbar buttons, or accelerators—also use the message-map mechanism. MFC defines a standard routing of command messages among the application, frame window, view, and document objects in your program. You can override this routing if you need to.

Message maps also supply a way to update user-interface objects (such as menus and toolbar buttons), enabling or disabling them to suit the current context.

MFC Fundamentals

MFC's strong suit is its fundamental support for programming for Microsoft Windows. The following programming areas are of common interest:

- Frame windows
- Documents
- Views of documents
- Multiple views
- Special view types, such as scroll views and form views
- Dialog boxes and property sheets
- Windows Common Controls
- Mapping Windows messages to handler functions
- Toolbars and other control bars
- Printing and print preview
- Serialization of data to and from files and other media
- Device contexts and GDI drawing objects
- Exception handling
- Collections of data objects
- Diagnostics
- Strings, rectangles, and points
- Date and time
- And considerably more

Appendix – B

MFC Document/View Architecture

The parts of the MFC framework most visible both to the user and to the developer, are the document and view. Most of the work in developing an application with the framework goes into writing document and view classes. This article describes, the purposes of documents and views and how they interact in the framework.

The CDocument class provides the basic functionality for programmer-defined document classes. A document represents the unit of data that the user typically opens with the Open command on the File menu and saves with the Save command on the File menu[4],[8].

The CView class provides the basic functionality for programmer-defined view classes. A view is attached to a document and acts as an intermediary between the document and the user: the view renders an image of the document on the screen and interprets user input as operations upon the document. The view also renders the image for both printing and print preview.

The following figure shows the relationship between a document and its view.

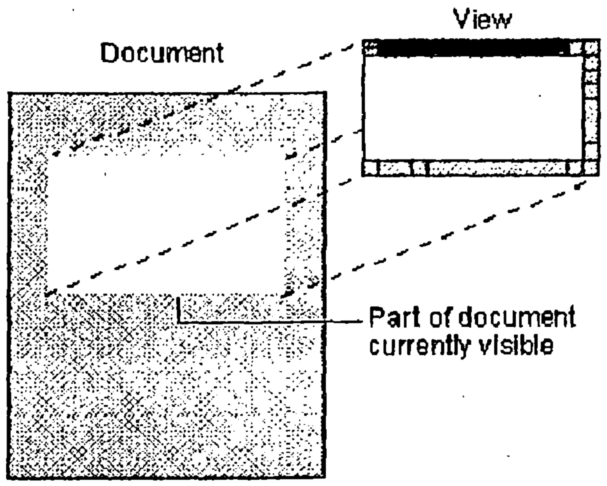
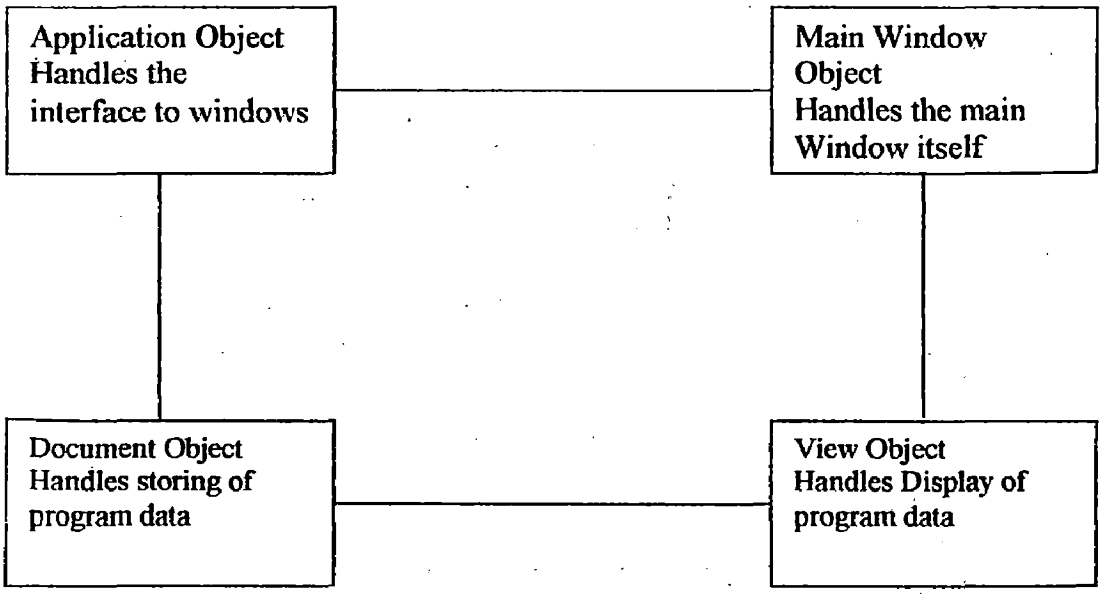


Figure b(I) Document and View

The document/view implementation in the class library separates the data itself from its display and from user operations on the data. All changes to the data are managed through the document class. The view calls this interface to access and update the data.

A document template creates documents, their associated views, and the frame windows that frame the views. The document template is responsible for creating and managing all documents of one document type.

A Portrait of the Document/View Architecture



Application Object Handles the interface to windows
Main Window Object Handles the main Window itself
Document Object Handles storing of program data
View Object Handles Display of program data

Figure b(ii) A portrait of Document/View Architecture

Documents and views are paired in a typical MFC application. Data is stored in the document, but the view has privileged access to the data. The separation of document from view separates the storage and maintenance of data from its display.

Gaining Access to Document Data from the View

The view accesses its document's data either with the GetDocument function, which returns a pointer to the document or by making the view class a C++ friend of the document class. The view then uses its access to the data to obtain the data when it is ready to draw or otherwise manipulate it. For example, from the view's OnDraw member function, the view uses GetDocument to obtain a document pointer. Then it uses that pointer to access a Cstring data member in the document. The view passes the string to the TextOut function.

User Input to the View

The view might also interpret a mouse click within itself as either selection or editing of data. Similarly it might interpret keystrokes as data entry or editing. Suppose the user types a string in a view that manages text. The view obtains a pointer to the document and uses the pointer to pass the new data to the document, which stores it in some data structure.

Updating Multiple Views of the Same Document

In an application with multiple views of the same document – such as a splitter window in a text editor – the view first passes the new data to the document. Then it calls the document's UpdateAllViews member function, which tells all views of the document to update themselves, reflecting the new data. This synchronizes the views.

Dialog Bars

Control bars greatly enhance a program's usability by providing quick, one-step command actions. Class **CcontrolBar** provides the common functionality of all toolbars, status bars, and dialog bars. CcontrolBar provides the functionality for positioning the control bar in its parent frame window. Because a control bar is usually a child window of a parent frame window, it is a "sibling" to the client view or MDI client of the frame window. A control-bar object uses information about its parent window's client rectangle to position itself. Then it alters the parent's remaining client-window rectangle so that the client view or MDI client window fills the rest of the client window.

A dialog bar is a control bar, based on a dialog-template resource, with the functionality of a modeless dialog box. Dialog bars can contain any Windows control. As in a dialog box, the user can tab among the controls. Dialog bars can be aligned to the top, bottom, left, or right side of a frame window. Dialog bars are control bars with both toolbar and dialog-box characteristics. They behave like toolbars, but because they are based on dialog templates, they can have any control that a dialog box can. MFC supports dialog bars with class **CdialogBar**.

There are several key differences between a toolbar and a CdialogBar object. A CdialogBar object is created from a dialog-template resource, which we can create with the Visual C++ dialog editor and which can contain any kind of Windows control. The user can tab from control to control. And we can specify an alignment style to align the dialog bar with any part of the parent frame window or even to leave it in place if the parent is resized.

While it is normal to derive our own dialog classes from **Cdialog,** we do not typically derive our own class for a dialog bar. Dialog bars are extensions to a main window and any dialog-bar control-notification messages, such as **BN_CLICKED** or **EN_CHANGE,** will be sent to the parent of the dialog bar — the main window.

Dialog Boxes

Creating a dialog object is a two-phase operation. First, construct the dialog object, then create the dialog window. Modal and modeless dialog boxes differ somewhat in the process used to create and display them. During the life cycle of a dialog box, the user invokes the dialog box, typically inside a command handler that creates and initializes the dialog object, the user interacts with the dialog box, and the dialog box closes. The following table lists how modal and modeless dialog boxes are normally constructed and displayed.

Dialog Creation

Dialog type	**How to create it**
Modeless	Construct **Cdialog,** then call **Create** member function.
Modal	Construct **Cdialog,** then call **DoModal** member function.

Table 1

Creating Modeless Dialog Boxes

For a modeless dialog box, we must provide our own public constructor in our dialog class. To create a modeless dialog box, call our public constructor and then call the dialog object's **Create** member function to load the dialog resource. We can call Create either during or after the constructor call. If the dialog resource has the property **WS_VISIBLE,** the dialog box appears immediately. If not, we must call its **ShowWindow** member function.

For modeless dialog boxes, we might often extract data from the dialog object while the dialog box is still visible. At some point, the dialog object is destroyed; when this happens depends on our code.

Creating Modal Dialog Boxes

To create a modal dialog box, call either of the two public constructors declared in **Cdialog**. Next, call the dialog object's **DoModal** member function to display the dialog box and manage interaction with it until the user chooses OK or Cancel. This management by DoModal is what makes the dialog box modal. For modal dialog boxes, DoModal loads the dialog resource.

For modal dialog boxes, our handler gathers any data the user entered once the dialog box closes. Since the dialog object exists after its dialog window has closed, we can simply use the member variables of our dialog class to extract the data.

Initializing the Dialog Box

After the dialog box and all of its controls are created but just before the dialog box (of either type) appears on the screen, the dialog object's **OnInitDialog** member function is called. For a modal dialog box, this occurs during the **DoModal** call. For a modeless dialog box, **OnInitDialog** is called when **Create** is called. We typically override OnInitDialog to initialize the dialog box's controls, such as setting the initial text of an edit box. We must call the OnInitDialog member function of the base class, **Cdialog**, from our OnInitDialog override.

Retrieving Data from the Dialog Object

Dialog data exchange (**DDX**) lets us exchange data between the controls in the dialog box and member variables in the dialog object more easily. This exchange works both ways. Dialog data exchange (DDX) is an easy way to initialize the controls in your dialog box and to gather data input by the user. To initialize the controls in the dialog box, we can set the values of data members in the dialog object, and the framework will transfer the values to the controls before the dialog box is displayed. Then we can at any time update the dialog data members with data entered by the user. At that point, we can use the data by referring to the data member variables. To use DDX, we define member variables in the dialog box, form view, or record view class, and associate each of them with a dialog box control. The framework transfers any initial values to the controls when the dialog box is displayed. When we click OK, it updates the variables with the data that we entered. To use the DDX mechanism, we have to set the initial values of the dialog object's member variables, typically in our **OnInitDialog** handler or the dialog constructor. Immediately before the dialog is displayed, the framework's DDX mechanism transfers the values of the member variables to the controls in the dialog box, where they appear when the dialog box itself appears in response to **DoModal** or **Create**. The default implementation of *OnInitDialog* in *Cdialog* calls the **UpdateData** member function of class *CWnd* to initialize the controls in the dialog box.

The same mechanism transfers values from the controls to the member variables when the user clicks the OK button (or whenever we call the **UpdateData** member function with the argument **TRUE**).

The following figure illustrates dialog data exchange.

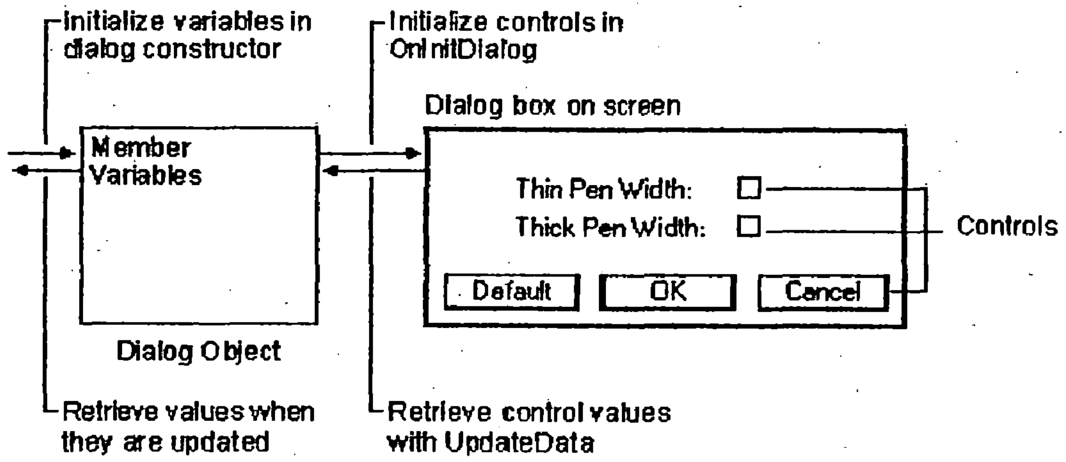

Initialize variables in dialog constructor
Initialize controls in OnInitDialog
Dialog box on screen
Member Variables
Thin Pen Width:
Thick Pen Width:
Controls
Default
OK
Cancel
Dialog Object
Retrieve values when they are updated
Retrieve control values with UpdateData

Figure b(iii) Dialog Data Exchange

UpdateData works in both directions, as specified by the **BOOL** parameter passed to it. To carry out the exchange, *UpdateData* sets up a *CDataExchange* object and calls our dialog class's override of *CDialog's DoDataExchange* member function. **DoDataExchange** takes an argument of type *CDataExchange*. The **CDataExchange** object passed to **UpdateData** represents the context of the exchange, defining such information as the direction of the exchange.

When we (or ClassWizard) override **DoDataExchange**, we have to specify a call to one DDX function per data member (control). Each DDX function knows how to exchange data in both directions based on the context supplied by the *CDataExchange* argument passed to our DoDataExchange by **UpdateData**.

We can also arrange for the values of dialog controls to be validated automatically with dialog data validation (**DDV**). Dialog data validation (DDV) is an easy way to validate data entry in a dialog box. To take advantage of DDX and DDV in our dialog boxes, use ClassWizard to create the data members and set their data types and specify validation rules. With DDV, dialog box information entered by the user is validated automatically. We can set the validation boundaries: the maximum length for string values in an edit-box control or the minimum or maximum numeric values when we expect a number to be entered. The DDV function typically alerts the user with a message box if the validation fails and puts the focus on the offending control so the user can reenter the data.

DDV Variable Types

Variable type	**Data validation**
Cstring	Maximum length
Numeric **(int, UINT, long, DWORD, float, double)**	Minimum value, maximum value

Table 2

We can define the maximum length for a **CString** DDX variable or the minimum or maximum values for a numeric DDX variable at the time we created it.

At run time, if the value entered by the user exceeds the range we specified, the framework automatically displays a message box asking the user to reenter the value. The validation of DDX variables takes place all at once when the user clicks OK to accept the entries in the dialog box.

For a modal dialog box, we can retrieve any data the user entered when **DoModal** returns **IDOK** but before the dialog object is destroyed. For a modeless dialog box, we can retrieve data from the dialog object at any time by calling **UpdateData** with the argument **TRUE** and then accessing dialog class member variables.

Closing the Dialog Box

A modal dialog box closes when the user chooses one of its buttons, typically the OK button or the Cancel button. Choosing the OK or Cancel button causes Windows to send the dialog object a **BN_CLICKED** control-notification message with the button's ID, either **IDOK** or **IDCANCEL**. **CDialog** provides default handler functions for these messages: **OnOK** and **OnCancel**.

SDI (Single Document Interface)

When an application runs under Microsoft Windows, the user interacts with documents displayed in frame windows. A document frame window has two major components: the frame and the contents that it frames. A document frame window can be a single document interface (SDI) frame window or a multiple document interface (MDI) child window. Windows manages most of the user's interaction with the frame window: moving and resizing the window, closing it, and minimizing and maximizing it. We can manage the contents inside the frame.

Frame Windows and Views

The MFC framework uses frame windows to contain views. The two components—frame and contents—are represented and managed by two different classes in MFC. A frame-window class manages the frame, and a view class manages the contents. The view window is a child of the frame window. Drawing and other user interaction with the document take place in the view's client area, not the frame window's client area. The frame window provides a visible frame around a view, complete with a caption bar and standard window controls such as a control menu, buttons to minimize and maximize the window, and controls for resizing the window. The "contents" consist of the window's client area, which is fully occupied by a child window—the view. The following figure shows the relationship between a frame window and a view.

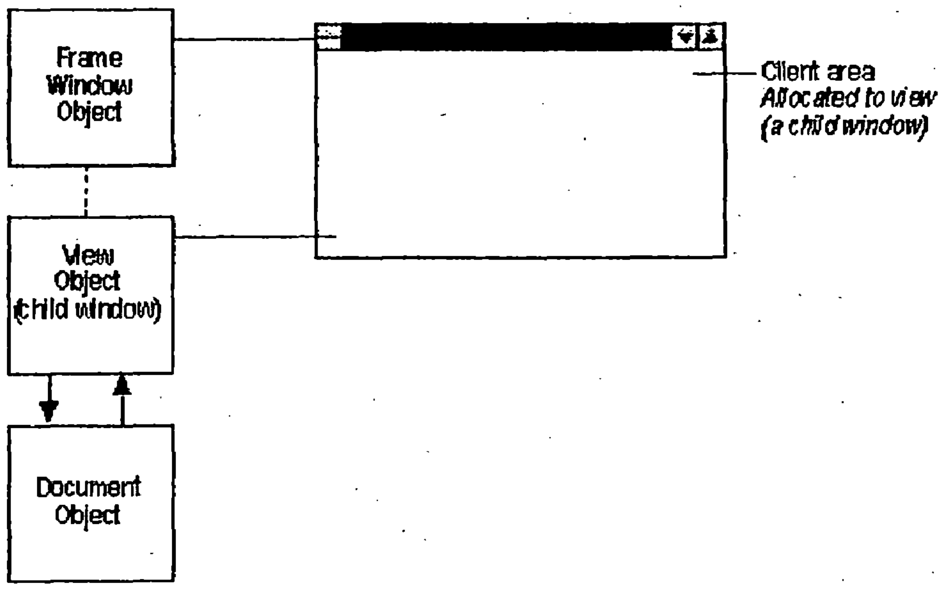


Figure b(iv) Relation between Frame Window and View